

ESKİŞEHİR OSMANGAZİ ÜNİVERSİTESİ
MÜHENDİSLİK MİMARLIK FAKÜLTESİ
MAKİNE MÜHENDİSLİĞİ BÖLÜMÜ

Mühendisler için FORTRAN 90/95 Programlama Dili ve Teknikleri

Prof. Dr. Zekeriya ALTAÇ
Yrd. Doç. Dr. Mesut TEKKALMAZ

ESKİŞEHİR OSMANGAZİ ÜNİVERSİTESİ
MÜHENDİSLİK MİMARLIK FAKÜLTESİ
MAKİNE MÜHENDİSLİĞİ BÖLÜMÜ

2005
ESKİŞEHİR



Mühendisler için FORTRAN 90/95 Programlama Dili ve Teknikleri

Prof. Dr. Zekeriya ALTAÇ
Yrd. Doç. Dr. Mesut TEKKALMAZ

T.C.
Eskişehir Osmangazi Üniversitesi
Mühendislik Mimarlık Fakültesi
Makine Mühendisliği Bölümü
Eskişehir

Eylül 2005

MÜHENDİSLER İÇİN
FORTRAN PROGRAMLAMA
1. Basım
Osmangazi Üniversitesi,
Mühendislik Mimarlık Fakültesi
Ekim 1995, Batı-Meşelik, Eskişehir.

Osmangazi Üniversitesi, Yönetim Kurulunun 09.02.1995 gün ve 43 sayılı kararlı ile yazarın söz konusu kitabı kendi imkanları ile basmasıiçin gerekli izin verilmiştir.

MÜHENDİSLER İÇİN
FORTRAN 90/95 PROGRAMLAMA
DİLİ VE TEKNİKLERİ
Genişetilmiş 2. Basım
Eskişehir Osmangazi Üniversitesi,
Mühendislik Mimarlık Fakültesi
Ekim 2005, Batı-Meşelik, Eskişehir.

Basım:
Eskişehir Osmangazi Üniversitesi
Mühendislik Mimarlık Fakültesi
Koruma ve Yaşatma Derneği
Batı-Meşelik, Eskişehir.

İÇİNDEKİLER

ÖNSÖZ

Bilgisayarlar ve bilgisayar teknolojilerinin hızlı bir şekilde gelişimiyle birlikte bilgisayarların mühendislik ve hizmet sektörlerinde artarak kullanıldığına şahit olmaktayız. Bu gelişim beraberinde ofis ve mühendislik paket programlarının geliştirilmesine neden olmuştur. Matematiksel işlemlerin yapılmasına olanak sağlayan MATHEMATICA, MATLAB, MATCAD, MAPLE gibi programlar gelişmiş ve yaygınlaşmış; optimizasyon, diferansiyel denklem takımlarının çözümü gibi nispeten zor olan mühendislik problemlerinin çözümünde kullanılmaya başlanmıştır. Temel mühendislik hizmetleri için geliştirilen çeşitli çizim ve/veya hesap programları mühendislerin işlerini kolaylaştırmış olsa da, orta ve ileri düzeydeki problemlerin çözümünde, program yazma ihtiyacını henüz ortadan kaldırmamıştır. Halen kullanılan programlama dilleri, mühendislik dallarındaki ihtiyaçlara göre çeşitlilik göstermektedir; örneğin, bilgisayar yazılımları oluşturmak için C++, yapay zeka programcıları için PROLOG, üretim makinelerinin programlanması için PLC, bilgisayar donanımlarını programlayanlar için ASSEMBLER, web sayfası tasarımcıları için HTML ve JAVA vs kullanılır.

Öğrenciler sıklıkla "*modası geçmiş bir dil*" olan "*neden* FORTRAN'ı *öğreniyoruz*?" sorusunu sormaktalar. FORTRAN, adından da anlaşılacağı üzere (İngilizce de FORmula TRANslation kelimelerinin kısaltılıdır) mühendislik, fen bilimleri ve bilimsel çalışmaları için tasarlanmış bir programlama dilidir; ticari yazılım yazmak için (modelleme ve simülasyon programları hariç) kullanılan bir dil değildir. Piyasada sadece "programcı" olarak çalışanlar, bu nedenle, Fortran diline ihtiyaç duymazlar. Ancak Fortran halen mühendislik ve bilimsel çalışmalarında vazgeçilmez programlama dillerinden biri olarak yerini korumaktadır. Ayrıca birçok fen ve mühendislik probleminin çözümü için ortaya atılan program ve alt programların neredeyse tamamı FORTRAN dilinde kodlanmıştır; bu algoritmaları içeren programlar piyasada çeşitli adlar altında hazırlanan paketler şeklinde hizmete sunulmaktadır. Bazı nispeten basit programları diğer programlama dillerinde yazmak mümkündür; ancak FORTRAN dilinin fonksiyonelliği ve elverişliğini henüz geçmemişlerdir.

Bu kitap esasen Osmangazi Üniversitesi, Mühendislik Mimarlık Fakültesi, Makine Mühendisliği bölümünde okutulmakta olan Bilgisayar Programlama dersine yardımcı olmak için hazırlanmıştır. Bu kitabı hazırlarken ana hedefimiz sadece FORTRAN programlama dili ve deyimlerini öğretmek değil, aynı zamanda yapısal programlama mantığı ile temel mühendislik bilimleri uygulamasında gerekli sayısal tekniklere bir giriş yapmak olmuştur. Bu nedenle türev, integral, matrisler gibi konular işlenmiş, gerekli programlar temin edilmiştir.

Bu kitabın öğrencilerimizin (elbette gelecekteki mezunlarımızın) meslek yaşamlarında ihtiyaçları olan programlama bilgisine katkısı olmasını umuyoruz.

Prof. Dr. Zekeriya ALTAÇ Yrd. Doç. Dr. Mesut TEKKALMAZ

19 Eylül 2005

BÖLÜM 1
GENEL BİLGİLER

1.1 BİLGİ TOPLUMU

Günümüzde bilgisayarların üretim sektörünü, iş ve özel hayatımızı nasıl etkilediğini her gün görmekte ve yaşamaktayız. Her gün, her saat bankalarda, okullarda, özel/tüzel kurum ve kuruluşlarda, otomasyon üretim makinelerinde vb yerlerde kullanılmakta, kısacası her yerde ve her şeyden sürekli veri toplanmaktadır. Fakat bu veriler ilk bakışta düzensiz ve anlamsız şekilde olmaktadır. Tüm bu veriler faydalı veriler haline dönüştürülebilir. Bu işleme *Veri işleme Sistemi* denir.

Veri işleme sistemleri, bilgisayar kullansın veya kullanmasın daima bilgilerin *girişi*, *depolanması*, *işlenmesi*, *çıktı alınması* ve *kontrolü* olmak üzere beş ana bölümden oluşur. Tüm bu bölümlerde verilerin sistemle ilişkisi sağlanır. Daha sonra da veriler, kurumların, toplumun ve kişilerin istediği şekle veya formata dönüştürülür.

Tarihin başlangıcından beri insanlık veri işleme sistemlerine ihtiyaç göstermiştir; insanoğlu sayma gereği duyduğunda ilk olarak parmaklarını kullanma yoluna gitmiş, sayıların toplanması amacıyla düğümlü ipler veya daha sonra Çinliler ve eski Yunanlılar "abaküs" adı verilen aracı, aritmetik işlem yapmakta kullanmışlardır. İlk eldeli toplama işlemini yapabilen hesap makinası 1642'de Fransız Blaise Pascal tarafından icat edilmiştir. Bugün bunların daha gelişmiş olanları (artık ülkemizde de kullanım dışı kalan) FACID hesap makineleridir. 20. asırdan önce veri işlemeleri genelde mekanik cihaz ve aletlerle yapılırdı. Bu yüzyılın başında Dr. Herman Hollerith adında bir bilim adamı 1890 yılında A.B.D. de yapılan nüfus sayımı sonuçlarını değerlendirmek için delikli kartlar üzerine işlenen bir yöntem ve elektro-mekanik bir cihaz tasarlamıştır. Daha sonra 1940'larda, bu makineler özellikle askeri alanda kullanılmak üzere yapılan bilimsel araştırmalar yardımıyla geliştirilmiş ve sonuçta elektronik bilgisayarların geliştirilmesinde öncü rol oynamışlardır.

Otomatik hesaplayıcılara elektroniğin uygulaması 1946 yılında J. P. Eckert ve J. W. Mauhly tarafından ENIAC'ın (**E**lectronic **N**umerical **I**ntegrator **A**nd **C**alculator) yapımı ile gerçekleşti. Bu makine 30 ton ağırlığındaydı ve 135 metre karelik bir odayı doldurmaktaydı. Bir kişinin masa üstü hesap makineleriyle 20 saatte yaptığı işleri, bu makine 30 saniyede yapabilmekteydi. Yaklaşık 20 bin radyo lambası kullanmakta ve 150 kW enerji kullanmaktaydı. 1964-1970 arasında, üretilen bilgisayarlarda entegre devreler kullanılmaya başlandı. On binlerce devre küçük bir silikon yongaya yerleştirildi. Düşük maliyet, yüksek güvenirlilik, çok daha ufak boyutlarda üretilebilmesi, düşük enerji harcaması ve hızlı olması yongaların mikro-bilgisayar yapımında kullanılmasına temel neden oldu.

İlk popüler grafiksel işletim sistemi 1984 yılında, Apple Macintosh ile piyasaya girdi. Microsoft firması Macintosh için sözlük işlemci ve elektronik tablo programı yazdı. İlk IBM Kişisel Bilgisayarı, 1981 yılının Ağustos ayında pazara çıkardı. IBM, 1983 baharında, şirketin, içinde sabit disk bulunan ilk kişisel bilgisayarı olan PC/XT'sini piyasaya sürdü. Disk, yerleşik bir depolama aygıtı olarak çalışıp, 10 MB idi. 1984'te, IBM, Intel'in 80286 mikro işlemcisine dayalı, PC AT adlı yüksek performanslı ikinci kuşak bilgisayarını tanıttı. IBM PC'den üç kat

hızlıydı. 1986 yılında ilk diz üstü bilgisayar olan PC convertible piyasaya sürüldü; akü ile çalışmakta olup ağırlığı 6 kg idi. 1989 yılında IBM dönemin en güçlü bilgisayarı olan 486/25MHz Power Platform'unu tanıttı. Cihazın sabit diski 30 MB'a yükselmişti. Bilgisayarlar 1980'li yıllarda DOS operatör sistemi ile çalışmaktaydı. 1990 Mayıs ayında, Windows 3.0 piyasaya sürüldü. Windows'un eski versiyonları mevcut olmasına rağmen pek kullanılışlı değildi.

Günümüzde artık bilgisayarlar iş ve endüstri alanında geniş ölçüde kullanılmaktadır. Ayrıca kişisel bilgisayarların da kullanımı artmakta ve yukarıda bahsedilen alanlardan başka, Internet aracılığıyla alışveriş (e-alışveriş), bankacılık (e-bankacılık), haberleşme (e-posta), başta olmak üzere, artık evlerdeki kullanımı her geçen gün hızla yaygınlaşmaktadır. Bütün bunların sonucu olarak toplumlar, endüstri toplumundan çıkıp, endüstri sonrası olarak düşünülen bir *bilgi toplumu* haline gelmiştir.

1.2 BİLGİSAYAR SİSTEMLERİ

1.2.1 BİLGİSAYAR TİPLERİ

Günümüzde kullanılan 2 tür bilgisayar vardır: analog ve sayısal (digital) bilgisayarlar. Analog bilgisayarlar fiziksel sistemlerin simülasyonunda kullanılır. Bir analog bilgisayarın çalışma sistemi zamanla sürekli (kesikli olmayan) değişen ölçüm sinyalleri ile ilgilidir. Bu yüzden analog bilgisayarlar genellikle sürekli ölçüm ve kontrol gerektiren alanlarda kullanılır. Daha ziyade, araştırma geliştirme ve üretim sektörlerinde kullanılmışlardır; halen kullanımı olmasına rağmen bilgisayar donanımların hızla gelişmesi analog bilgisayarların kullanım alanlarını daraltmıştır.

Sayısal bilgisayarlar, analog bilgisayarlara karşın zamanla kesikli şekilde değişen büyüklüklerle çalışır. Bu tip bilgisayarlar ölçmekten ziyade sayar ve fiziksel ölçümler yerine sayılar kullanır. İşletmelerde kullanılan veriler kesikli (yani belirli aralıklarla toplanan veriler) olduğundan, bu alanlarda da sayısal bilgisayarlar kullanılması doğaldır.

Yukarıda açıklanan her iki sistemi de içeren cihazlara hibrid (hybrid) bilgisayarlar adı verilir. Birçok ticari, bilimsel ve endüstriyel bilgisayar sistemleri analog ve sayısal cihazların kombinasyonundan meydana gelir. Bu tip cihazlar günümüzde gelişmekte ve her alanda kullanılmaktadır.

1.2.2 BİLGİSAYAR ELEMANLARI

Bir bilgisayar sistemindeki gözle görünen fiziksel parçalara *donanım*, sistemin gözle gözükmeyen ve yapması gereken işlevleri yerine getiren bilgisayar programlarına da *yazılım*, bilgisayar sistemindeki yongalara üretildikleri şirketlerde konulan programlara da *bellenim* adı verilir.

Bir bilgisayar sistem yapısı, *ana bellek* (depolama) birimi etrafında organize edilmiştir. Çünkü bilgisayar tarafından kullanılan tüm veri ve komutlar işlenmeden önce ana bellekten geçer. Buradan yola çıkarak sayısal bir bilgisayarın fonksiyonel birimleri şöyle sıralanabilir: (1) *Ana ve ikincil bellek birimleri*, (2) *Merkezi işlem birimi*, ve (3) *Girdi ve çıktı birimleri*.

1.2.3 ANA VE İKİNCİL BELLEK BİRİMLERİ

Ayrıca iç bellek olarak adlandırılan ana bellek birimi, merkezi işlem birimi ile aynı yerde bulunur. Bu birimin amacı:

1. İşlenecek tüm verileri tutmak,
2. Ara işlem sonuçlarını tutmak,
3. Elde edilen sonuçları çıktı cihazına göndermeden önce tutmak,
4. Devam etmekte olan işlem için gerekli tüm komutları tutmak.

Ana bellek, iki çeşit bellek ortamından oluşur: *magnetik-çekirdek* bellek ve *yarı iletken* (semi-conductor) bellek. Bunlardan birincisi olan magnetik çekirdek ortamı devredeki güç kesildiğinde içindeki bilgileri kaybetmeyen (non-volatile) bir bellek ortamıdır. Yarı iletken bellek ise, güç kesildiğinde içinde tuttuğu içeriği kaybeden, içindeki veriye gelişigüzel bir şekilde erişilebilen bir bellek olduğundan bu çeşit belleğe RAM (***R****andom* ***A****ccess* ***M****emory* **-** *Rasgele Erişimli Bellek*) de denir. İşlem sırasında RAM'deki içerik sürekli değişir.

Daha önce yukarıdaki kısımlarda bahsedilen sistemin çalışması için gerekli (özellikle bellenim) programların depolandığı güç kesilmesinde silinmeyen belleğe de kısaca ROM (***R****ead* ***O****nly* ***M****emory* **-** *Salt Oku Belleği*) denir.

Son zamanlarda, ROM olarak kullanılan fakat istenildiğinde silinip, tekrar programlanabilen bellekler de vardır. Bunlara EPROM (***E****raseable* ***P****rogramable* ***R****ead* ***O****nly* ***M****emory* **-** *Silinebilir Programlanabilir Salt Oku Belleği*) denir. Bunlar gereğinde çeşitli yöntemlerle tekrar programlanabilen fakat ROM gibi çalışan bellek birimleridir. İkincil bellek birimi olarak yıllardan beri elektro-mekanik ortam kullanımdadır. Bu ortam, bilgisayar sistemi dışında da bilgi depolanması için kullanılan ve istenildiği gibi silinip değiştirilebilen bir depolama sistemidir. Bunlar manyetik ve optik diskler, disketler ve CD ve kasetlerdir. Kütlesel olarak daha fazla bilgi depolanması için kullanılan başka cihazlar da vardır. Magnetik teypler, daha çok ana ve süper bilgisayarlarda bilgi depolama amacı ile kullanılır. CD, disk ve disketler ise kişisel bilgisayarlarda oldukça pratik şekilde kullanılabilen ve birçok boyutta ve kapasitede olan depolama ortamlarıdır. Kişisel bilgisayarlarda kullanılan disklere *sabit disk* (Fixed Disk veya Hard Disk), disketlere de esnek disket (Floppy diskette) denir. Tekrar tekrar kullanılabilen/yazılabilen (re-writeable) veya bir kez kullanılabilen, 700 MB gibi, oldukça fazla miktarda bilgi depolayabilen CD'ler de son yıllarda bilgi depolamada kullanılan araçlar haline gelmiştir.

Son yıllarda bellek çubukları ve cepte taşınabilir sabit diskler yaygın olarak kullanıma girmiştir. 1 MB'lık bellek çubuklarının yakın zamanda kapasitelerinin daha da artacağı tahmin edilmektedir. Taşınabilir sabit disklerin kapasitesi ile fiyatı artmaktadır. Ancak yakın zamanda bu alanda da hem kapasitesinin daha da artması hem de fiyatlarında düşüşlerin yaşanması beklenmektedir.

1.2.4 MERKEZİ İŞLEM BİRİMİ

Merkezi işlem Birimi, bilgisayarın beyni olup iki parçadan oluşmuştur: kontrol birimi (***C****ontrol* ***U****nit*-CU) ve aritmetik mantık birimi (***A****rithmetic* ***L****ogic* ***U****nit*-ALU).

Kontrol birimi, tüm bilgisayar sistemini kontrol ve organize eden birimdir. Ana bellekte depolanmış programdan gerekli komutları alır, yorumlar ve komutları yerine getirmek için

gerekli sinyalleri sistemin diğer birimlerine gönderir.

Her bilgisayar sisteminde gerçekleşmesi amaçlanan fonksiyonların her biri, örneğin dört işlem (toplama, çıkarma, çarpma ve bölme) ve karşılaştırma işlemi (küçük, büyük, eşit vs), için ayrı bir devre hazırlanmıştır. Bu devrelerin her birine tasarlayıcı tarafından bilinen, birbirlerinden farklı olan ve işlem kodu olarak adlandırılan numaralar verilir. Bu nedenle bilgisayar türlerine göre aynı işlem için bile bu kodlar da farklılık görülebilir. Bir bilgisayar programı içindeki her komut, işlem kodu ile birlikte işlemin üzerinde yapılacağı bilginin bellek konumunun adresini de taşır. Buna göre; kontrol birimi, komut sayıcı tarafından belirlenen konumda bulunan komutu okur ve işlem sırası gelen komutun adresini belirler; bellekten elektrik uyarısı halinde ulaştırılan komut içinde adresini verdiği bilgi üzerinde kontrol birimlerinin işlem kodu ile belirtilen işlemin yapılmasını sağlar ve bir sonraki komutu ve adresini belirleyerek işleme devam eder.

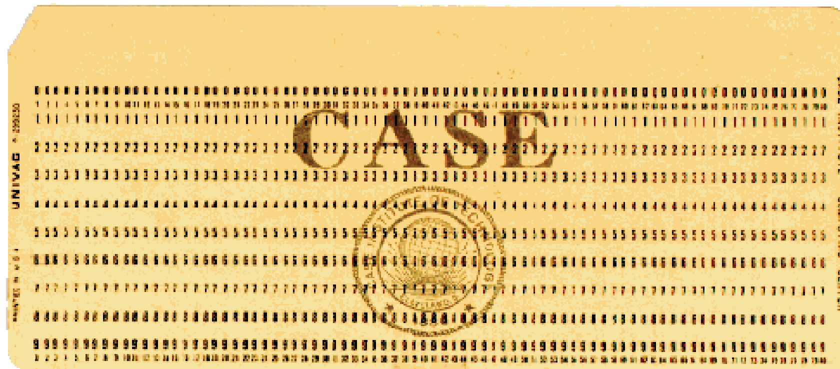
Aritmetik mantık birimi, kontrol birimi tarafından, ana bellekten alınan emirleri yerine getirir ve sonuçları ana belleğe geri gönderir. Burada yerine getirilen komutlar genelde matematiksel işlemlerdir.

Bilgisayar sistemlerinin tipi (örneğin kişisel bilgisayarların tipleri 486, Pentium, Triton, AMD, Cyrix v.s.) ve hızları (200 MHz'den başlayan ve her geçen gün hızları artmaktadır, v.s.) bu birimin tipi ve hızları ile belirtilmektedir.

1.2.5 GİRDİ VE ÇIKTI BİRİMLERİ

Bilgisayar bulunduğu ortamla mutlaka haberleşmek zorundadır. Girdi birimleri; verileri, programları ve diğer bilgileri ana belleğe vermekte kullanılır. Çıktı birimleri ise kullanıcıya gerekli bilgileri sunmak için ana bellekten verileri almak veya daha sonra kullanılmak üzere verileri ikincil bir bellek ortamında saklamak için kullanılır. Bazı cihazlar aynı anda hem girdi hem de çıktı cihazı olarak kullanılır.

1950-1980'li yıllarda klavye ile bilgi girişi teknolojisi henüz gelişmemişti. Her program satırı Şekil 1'de görülen bir karta delinir; bu kartları okuyan özel cihazlar aracılığıyla şimdi klavyeden girdiğimiz komut ve açıklamalara dönüştürülürdü. Son yirmi yılda girdi cihazları gelişti. Girdi cihazları olarak, klavye, tarayıcılar (scanner), ışık kalemleri, barkod okuyucuları, fareler, optik okuyucular, sayısallaştırıcı (digitizer) tabletler, video ve kameralar, ses aygıtları v.s gibi cihazlar mevcuttur.



Şekil 1.1 Fortran programının bir satırı için kullanılan kart.

Çıktı cihazları olarak, monitörler (VDU - Visual Display Unit), yazıcılar, grafik çizicileri (plotter), hoparlörler v.s. gibi cihazlar kullanılmaktadır.

Ayrıca hem girdi hem de çıktı cihazı olarak manyetik teypler, disk ve disketler, bellek çubukları, taşınabilir sabit diskler kullanılmaktadır.

Tüm bunlardan başka ileri teknoloji ürünü olarak ses tanıma ve cevaplama cihazları da vardır. Bunlardan da bilgisayar sistemini ve diğer programları (özellikle kelime-işlem programlarını) çalıştırmak için günümüzde faydalanılmaktadır.

Yukarıda bahsedilen çıktı cihazlarından olan monitörlerdeki yeni gelişmelerle ekran *çözünürlüğü* (resolution), renk sayısı, hız ve kapasitesi (grafik gösterimi için) oldukça artırılmıştır. Ayrıca dokunmatik hale getirilip, klavye gibi kullanımı uygulamaları vardır. Bunlardan başka ekran tipi olarak CRT (***C**athode **R**ay **T**ube*), gaz plazma ve LCD (***L**iquid **C**rystal **D**isplay*) teknolojileri kullanılmaktadır. Tüm bu teknolojilerden kullanılan bilgisayar tipine bağlı olarak (PC-kişisel bilgisayar ve notebook-defter) faydalanılmaktadır.

Tabii ki kullanılan yazıcıların da birçok tipleri vardır. Bunlar arasında artık neredeyse demode olmuş nokta-matriksli (dot-matrix ve letter-quality), satır, sayfa, püskürtme tipli (ink-jet), lazer olanları (siyah-beyaz veya renki) vardır. Yazıcı hızları da bilgisayar sistemlerinin hızlarına ayak uydurmak için sürekli artırılmaktadır.

1.3 VERİLERİN BİLGİSAYARDA TEMSİLİ

1.3.1 İKİLİK SAYI SİSTEMLERİ

Bilgisayarlar, verileri hangi birim kullanılırsa kullanılsın, 0 ve l'in kombinasyonunu kullanarak ikilik düzende depolama yaparlar. Bilginin temel birimi "*bit*"tir. Bir *doğru-yanlış* veya *evet-hayır* veya *açık-kapalı* ile sağlanan bilgi miktarıdır. Yani 1 bit, bu iki durumdan sadece birini temsil eder. Sembolik olarak iki durumu temsil eden basamaklar 0 ve l'dir. Herhangi bir anda bir elektrik devresinden bir akım *geçebilir* (doğru) veya *geçmeyebilir* (yanlış). Akım bir tek yönde veya tersi yönde akabilir. Gerilim yüksek veya düşük olabilir; bir manyetik kaydedicinin bir unsuru manyetik olabilir veya olmayabilir. Bunların her biri iki elektriksel değerin bir alternatifi olup, bunlardan sadece birisi varolur.

Bitlerin en iyi uygulama alanlarından biri de ikilik sayı sistemidir ve sayılar 0'lar ve l'lerden oluşan bir dizi karakterden ibarettir. Bunun altında yatan esas Tablo 1.1 de verilmektedir.

Tablo 1.1: Onluk düzendeki sayıların ikilik düzendeki karşılığı.

Sayı	ikilik Değeri
0	0
1	1
2	10
4	100
8	1000
16	10000

Bu tablodan, ikilik değerin sonuna bir sıfır eklemenin onluk düzendeki sayının iki ile çarpımına karşılık geldiği görülmektedir. Bu tablo değerlerinin birleştirmesiyle onluk düzendeki diğer sayılar türetilebilir.

3=11 (2+1) 5=101 (4+1) 6=110 (4+2) 7=111 (4+3)

Bu örneklerden 0 ile 7 arasındaki sayılar için sadece 3 bit'e (000—111) ihtiyaç olduğu aşikardır. Daha büyük sayılar için bit gereksinimi artmaktadır. Bütün sayıların ve karakterlerin bit bazında kodlanmalarına standartlar getirilmiştir. Bunlardan EBCDIC (**E**xtended **B**inary **C**oded **D**ecimal **I**nterchange **C**ode-Genelleştirilmiş İkilik Kodlu Ondalık Değişim Kodu) karakter başına 8 bit kullanır. Bunun anlamı da 256 kombinasyonun mevcut olmasıdır. Sadece, 0 ile 9 arasındaki ilk 10 basamak için, ilk dört bit 1111 olarak atanır. Örneğin, bazı sayısal ve alfa sayısalların ikilik sistemdeki kodları 3=11110011, 7=11110111, A=11000001, Z=11101001 v.s olmaktadır.

Başka kodlama sistemleri de geliştirilmiştir. Bugün en yaygın olarak kullanılan kodlardan biri de ASCII (**A**merican **S**tandard **C**ode for **I**nformation **E**xchange - Amerikan Bilgi Değişimi Standard Kodu) olup, 7 bitlik bir koddur. Kodlamalardaki kargaşayı gidermek ve bir standartlaşmaya gitmek amacıyla bir araya gelen iletişim endüstrisinin ortaya koyduğu bir koddur. Bu kodlama sistemi 128 şablon sunmaktadır. Bunlardan bazıları 0=0110000, 1=0110001, 9=0111001, A=1000001, Z=1011010 v.s dir.

1.3.2 OKTAL VE HEKZADESİMAL SAYISAL DÜZEN

Sayılar büyüdükçe ikilik düzende sayıların temsili oldukça uzun ikilik-düzen sayıları ile sonuçlanır ve bu sayıların takibi güçleşir. Bu problemden sakınmak için ikilik-düzen sayılarının 3 veya 4 bitlik guruplara ayrıldığı 8-lik düzen (oktal) veya 16-lık düzen (hekzadesimal) tek bir sayı ile temsil edilebilir. Bu fikri daha iyi anlamak için ikilik düzende 0 (=000_2) ile 7 (=111_2) arasındaki herhangi bir sayıyı incelediğimizde bu sayıların 3 bit ile temsil edildiğini görürüz. Bu sayılar sekizlik (oktal) düzende kullanılan temel sayılardır. Bir oktal sistemin basamak değerleri, 0'dan 7'e kadar, sekiz değer alabilir. İkilik düzendeki herhangi bir sayı 3 bitlik guruplara ayrılabilir ve bu gurupların yerine sekizlik karşılıklarını kullanabiliriz; örneğin, 010001001100_2 sayısını 010 | 001 | 001 | 100_2 şeklinde temsil edebiliriz. Burada her üç bitlik gurup bir sekizlik sayıyı temsil eder ki bu sayıyı 2114_8 şeklinde yazabiliriz.

Benzer şekilde 0 (=0000_2) ile 15 (=1111_2) ile temsil edilen 4 bitlik guruplar 16'lık (hekza-desimal) düzeni temsil eder. Bir onaltılık sistemin basamakları 0'dan 9'a kadar on adet ve A'dan F'e kadar altı adet olmak üzere toplam 16 değer alabilir. Örneğin, 010001001100_2 sayısı 4 bitlik parçalara ayrıldığında 0100 | 0100 | 1100_2 elde edilir. Her 4 bitlik kısım onlatılık sayı düzenindeki karşılığı ile temsil edilirse $44C_{16}$ elde edilir.

1.4 BİLGİSAYAR DİLLERİ VE PROGRAMLAMA

Programlama dilleri iki ana kısımda toplanabilir: (1) makineye yönelmiş diller veya makine dilleri, (2) probleme yönelmiş diller ve bunlara ilişkin programlama sistemleri.

Makine dilinde, bilgisayarda kullanılan yongaların özellikleri göz önünde tutularak, komutlarla yapılacak işlemler dizisi biçiminde ifade edilir; sonra programcı tarafından bu işlemler dizisi, bilgisayarın makine dilinde (ASSEMBLY dili olarak da anılır) kodlanır. Düşük seviyeli diller arasında sayılır. Bir makine dili olmakla beraber programcılar tarafından donanımları

programlamak için kullanılır. Elde edilen makine dilindeki program, gerekli veriler ile beraber, bilgisayar tarafından doğrudan icra edilebilir. Ancak makine dili ile programlamada bazı güçlükler mevcuttur. Tüm komutların makine dilinde kodlanması, yongaların farklılığı, komutların makine tarafından icra edilecek sırada yazılması, adreslemenin yapılması zorunluluğu, tüm mantıksal ve program düzenlemenin programcı tarafından yapılması, programcının kullandığı bilgisayar donanımını çok iyi anlaşılmış olması gereği, bu güçlüklerin başında sayılabilir.

Tablo 1.2: 2, 8, 10, 16'lık sayı düzeni tablosu.

10'luk Düzen	*İkilik Düzen*	*Oktal (Sekizlik) Düzen*	*Hekzadesimal (16'lık) Düzen*
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Bilgisayarların problem çözme aracı olarak kullanılmaları için, bilgisayarlar ve kullanıcıları arasında bir iletişimin kurulması gerekir. Bilgisayarlar, bir anlamda, 0 ve l ile temsil edilen iki karakterden oluşan bir alfabeye dayanan, elektriğin diliyle konuşurlar. Bu *Makine Dilinin* detayları, bilgisayar tasarlandığında üretici firma tarafından belirlenir. Diğer taraftan, insanlar, doğal olarak, kendi ana dilleriyle konuşurlar. Bu iletişim engeli bir şekilde aşılmalıydı. İşte, bu iletişimin sağlanması işlemine *programlama* denir. Yıllar boyunca birçok programlama dilleri geliştirildi ve birçoğu da zamanla yok oldu. Kullanıcı adım adım ne yapılacağını detaylı talimatlarla makineye bildirir. Bu talimatların makine diline çevirisi, üretici firma tarafından geliştirilen ve bilgisayarın belleğine yerleştirilen programlar ile gerçekleştirilir. Bu tür çevirici programlara *derleyici* (compiler) adı verilir.

1.4.1 YÜKSEK SEVİYELİ BİLGİSAYAR DİLLERİ

Günümüzde ihtiyaca göre bilgisayarlar geliştikçe; bunların kullanılmasına, çalıştırılmasına ve ihtiyaç durumuna göre bilgisayar dilleri de gelişmektedir. Kullanılan programlama dilleri bilgisayarın belleğinde bulunan çevirici, derleyici veya benzer bir program ile makine diline çevrilir ve arzu edilen işlem yerine getirilir.

1959-1964 arasında üretilen bilgisayarlarda transistörler (10 bin adet) kullanıldı. COBOL (işletmecilik, muhasebe amaçlı), FORTRAN (bilimsel ve mühendislik amaçlı), ALGOL (yapay-zeka sistemleri amaçlı) yüksek düzeyli diller ve işletim sistemleri geliştirildi.

Zamanımızda kullanılan programlama dilleri arasında FORTRAN, BASIC, PASCAL, C, C+ gibi diller bulunmaktadır. Bu dillerden başka, özel amaçlı bilgisayarlarda kullanılan, askeri silah sistemleri ile uydu izleme ve takip gibi sistemlerde, sivil amaçlı üretim sistemlerine yönelik (CNC parça programlama gibi) diller de vardır. Yukarıda bahsedilen diller arasında amaçları bakımdan farklar olsa da kullanılan programlama mantığı açısından pek fark yoktur.

FORTRAN (FORmula TRANslation), adından da anlaşıldığı gibi, matematiksel hesapların (mühendislikteki olduğu gibi) yoğun olduğu dallarda kullanılır. FORTRAN dili ilk önce 1964'de geliştirilmiş olup ilk versiyonu FORTRAN IV olarak anılır. Yaklaşık 12 sene ufak tefek farklılıklardan oluşan birkaç versiyonu daha geliştirilmiştir. Ancak 1977 yılında toplanan FORTRAN Standartları Kurulu, yeni komutların ilave edildiği ve yapısal programlamanın geliştirildiği FORTRAN 77 dilinin temellerini ilan eder. Bilgisayarların ve donanımların ve yeni işletim sistemlerinin gösterdiği gelişim üzerine, bu gelişimlerin sunduğu olanaklardan en iyi şekilde yararlanmak, programlama gayretlerini azaltmak, program hızını artırmak amacıyla FORTRAN Standartları Kurulu 1990 senesinde FORTRAN 90 ve kısa bir süre sonra 1995 senesinde toplandığında da FORTRAN 95 dillerini ilan eder. Aynı nedenlerden ötürü, 2000 ve 2003 yıllarında FORTRAN diline yapılan eksiltme ve ilavelerle tekrar yenilenmiştir, yeni kurallar getirilmiştir. Ancak 2005 yılı itibarıyla FORTRAN 2003 programlarını derleyecek yazılımlar henüz piyasaya sürülmemiştir. FORTRAN 2000 ve 2003 ile girdi/çıktı, veri manipülasyonu, türetilebilen veri tipi oluşturma gibi özeliklere eklemeler yapılmıştır. Ayrıca C dilinde yazılan prosedür alt programlarıyla daha kolay arayüz'ler oluşturulması amaçlanmıştır.

BASIC ise genel amaçlı ve kullanımı kolay olan bir dildir. PASCAL, yine FORTRAN gibi, matematiksel hesaplarda ve genel amaçlı olarak kullanılan bir dil olup eski popülerliğini C diline kaptırmıştır. C+ ise bilimsel alanda ve genel amaçlı kullanılan oldukça gelişmiş bir dildir. Tüm bu dillere *yüksek seviyeli diller* denir. Bu programlama dillerinin Visual Basic, Visual Pascal ve Visual C/C++ olarak adlandırılan yazılımları mevcuttur.

Bu diller arasındaki fark (amaçlarından başka) bu dillerde kullanılan deyimlerdir. Bazılarında aynı tip deyimler aynı amaçlar için kullanılır. Ama çoğu zaman kullanılan deyimler farklıdır. Fakat kullanılan deyimler ve komutlar İngilizce dilinden alınmış *kelimeler* veya *kısaltmalardan* oluşur.

Bilgisayarlar geliştikçe kullanılan tüm bu diller geliştirilmekte veya teknoloji gelişimle beraber programlama ihtiyaçlarına daha iyi cevap veren yeni diller ortaya çıkması olasıdır.

1.4.2 İŞLETİM SİSTEMLERİ

Üretici firma ile bunlara hizmet veren firmalar makinenin etkin bir şekilde kullanılması için gerekli bir takım programlar hazırlarlar. Bunlar bilgisayardaki bilgilerin saklandığı yerlere ait kayıtları, tarih, boyut ve isimlerine göre tutması, istenen bilgilerin ekrandan veya yazıcıdan çıktısının alınması v.s gibi işlemlerde kullanılırlar. Bu programların tümüne birden *işletim sistemi* (operating system) adı verilir. Günümüzde kişisel bilgisayarlarda kullanılan en yaygın işletim sistemleri Windows (95/98, 2000, XP, ME, NT vs) veya Unix işletim sistemleridir. Süper bilgisayar veya main-frame olarak adlandırılan çok kullanıcılı devasa bilgisayarlarda farklı işletim sistemleri kullanılmaktadır.

1.4.3 BİLGİSAYAR YARDIMIYLA PROBLEM ÇÖZME

Bir bilgisayar aracılığıyla problem çözmek için öncelikle çözülecek problemin türüne uygun bir programlama dili seçilmelidir. Fen ve mühendislik bilimlerine en uygun programlama dili FORTRAN olmasının nedeni diğer programlama dillerinde üstünde fazla durulmayan ve bu bilim dallarında yaygın olarak kullanılan fonksiyonlar ve programlama komutlarının (matrislerle ilgili işlemler, maksimum, minimum, hiperbolik, trigonometrik, üstel ve logaritmik fonksiyonlar vb), kısacası *arşiv* programlarının FORTRAN derleyicilerinde mevcut olmasıdır. Örneğin, A ve B kompleks iki sayının çarpımını diğer dillerde yapmak için daha uzun süren bir dizi işlem yapmak gerekirken, FORTRAN ile bu işlem kısaca `A * B` şeklinde tanımlanabilmektedir. Bu nedenle kompleks analizin yoğunlukla kullanıldığı bir problemin çözümünde başka bir dilin kullanımı zor ve yoğun bir programlama gerektirecektir.

Problem çözmenin adımları aşağıdaki sırayla verilebilir:

1. Problemin tanımlanması (fiziksel sistemi idealleştiren bir matematiksel modele çevrilmesi ve matematiksel denklemlerin formülleştirilmesi),
2. Problemin çözümüne uygun bir algoritmanın tespiti ve detaylı bir akış şemasının hazırlanması (problemin çözümü için adım-adım talimatlar dizisinin çizimsel gösterimi),
3. Programın bir programlama dilinde kodlanması (kaynak programın hazırlanması),
4. Programın bilgisayara bir girdi birimi vasıtasıyla, genellikle klavye, girişi ve yedek birimde (diskette) depolanması,
5. Programın programlama diline uygun bir derleyici yazılım ile derlenmesi ve arşiv fonksiyonlarına bağlanarak gerekli mantıksal/matematiksel fonksiyonların ana programa aktarılması,
6. Programın test edilerek, istenen işlemleri yapıp yapmadığının kontrol edilmesi,
7. Program çalıştırılarak merkezi işlem biriminde, aritmetik ve kontrol birimleri ile depolama biriminin gerektikçe kullanımı,
8. Sonuçların bir çıktı birimine (ekrana veya yazıcıya veya diskete) aktarılması.

Bu sırayı takip etmek problem çözme zamanını kısaltır; aksi takdirde yapılan hataları tespit etmek için uzun ve sıkıcı bir süre program ile boğuşmak zorunda kalabilirsiniz. Kısacası çözeceğiniz problemi, programlamaya başlamadan önce nasıl çözeceğinizi çok iyi bilmeniz gerekir. Bu nedenle problem çözme teknikleri, genellikle, programlama tekniklerinden daha fazla önem kazanmaktadır.

1.4.4 PROGRAM DERLEME VE BAĞLAMA İŞLEMLERİ

Bilgisayar programı yazım aşamasında kullanımı arzu edilen dilde, örneğin FORTRAN, kodlanır. Kodlanan bu programa *kaynak programı* (source program) denir. Kişisel bilgisayarlardaki kütük kullanımlarında, kütük isimleri bir uzantı ile beraber kullanılır. Bu uzantı FORTRAN 77 ile yazılmış kaynak programlarında, `isim.FOR` şeklindedir. FORTRAN 90 ve 95 dillerinde ise sırasıyla `isim.F90` ve `isim.F95` şeklinde olmaktadır.

Bilgisayar, maalesef, hazırladığımız kaynak programının dilinden, yani bu kitapta verilen küçük büyük programlardan, anlamaz. Programı makinenin (bilgisayarın) anlayacağı bir dile çevirmek gerekir. Bu nedenle, üretici veya yan firmalar, herhangi bir dilde yazılan programları makine diline çeviren yazılımlar hazırlarlar. Bu tür yazılımlara derleyici (compiler) adı verilir ve bilgisayarla beraber gelen standart programlardan olmayıp ayrıca satın alınması gerekir.

Derleyiciden geçirilen kaynak programında programlama kurallarına uymayan veya yanlış kullanılan deyimler varsa, bunlar tespit edilir ve kullanıcı uyarılır. Bu tür hatalara derleme hataları (compilation errors) denir. Derleyicilerin kullanım kılavuzunda bu hataların ne anlama geldiğini belirten bir listesi mevcuttur. Program, derleme işleminden, mutlaka hatalardan arınmış olarak geçmelidir. Halen çeşitli firmaların piyasaya sürdüğü FORTRAN derleyicileri vardır. Bunlardan kişisel, ağ ortamlı Windows veya Unix işletim sistemi altında çalışan bilgisayarlara göre hazırlananlardan bir kaçı MicroSoft Fortran, Lahey Fortran ve Prospero PC Fortran'dır

Amerika Ulusal Standardlar Enstitüsü (American National Standards Institute-ANSI) FORTRAN dilini ve standart deyimlerim tanımlamıştır. ANSI standartlarına uyan programlar bütün FORTRAN derleyicilerinde derlenebilir.

Program derleme işleminden başarı ile geçildiğinde sadece makinenin anlayacağı bir kütük yaratılır. Bu kütüğe hedef program (OBJect program) denir ve kaynak programın ismini taşır; fakat uzantısı (`.OBJ`) farklıdır. Örneğin, yukarıda bahsedilen `MATRIS.F90` (Fortran 90) veya `MATRIS.F95` (Fortran 95) isimli kaynak program başarı ile derlendikten sonra oluşturulan hedef program `MATRIS.OBJ` ismini alır.

Programın derleme aşamasından geçmesi, programın çalıştırılması için, yeterli değildir. Programın kullanıldığı sistemde çalıştırılabilmesi ve programda kullanılan bazı arşiv fonksiyonlarının ana program ile bağlantısının sağlanması için bağlama (link) aşamasından geçirilmesi gerekir. Bu işlem DOS ortamında çalıştırılan derleyicilerde `LINK.EXE` programı ile yapılmaktadır. Fakat Windows işletim sistemi altında çalışan derleyici yazılımlarının, kendi bağlayıcılarını (linker) tasarladıkları ve bunları kullandıkları bilinmektedir. Bazı derleyiciler, derleme işlemi sonrası otomatik olarak bağlama işlemini yapar.

Bağlama işleminden geçirilen programın *`isim`*`.EXE` uzantısını taşıyan yeni bir versiyonu oluşturulur. Programın, işte, bu versiyonuna çalıştırılabilir program (EXEcutable program) denir. Programın `.EXE` versiyonu yaratıldıktan sonra, programın ikonuna fare ile iki kez tıklamakla veya DOS ortamından çalıştırılacaksa program isminin komut satırından girilmesi, bunu takiben RETURN veya ENTER tuşuna basılmasıyla, çalıştırılır.

ALIŞTIRMALAR

1.1 Bir kişisel bilgisayardaki (PC) girdi, çıktı ve işlem birimlerini tanımlayınız. Girdi birimi birden fazla olabilir mi?

1.2 Bir bilgisayarın sabit disk ve esnek disk unsurundan bahsederken ne kastediliyor? Aralarındaki farklar nelerdir?

1.3 Bir PC için yazılım, donanım ve bellenim unsurlarına örnekler veriniz.

1.4 ANSI ve ASCII terimlerini tanımlayınız. Bu iki kısaltma arasında bir yakınlık var mıdır?

1.5 FORTRAN, PASCAL, C, COBOL ve BASIC dilleri hangi alanlarda kullanılmaları için tasarlanmışlardır?

1.6 Bir FORTRAN kaynak programının çalıştırılabilmesi için hangi aşamalardan geçirilmelidir? Bu aşamaları ayrıntılı olarak açıklayınız.

1.7 Aşağıda ikilik düzende verilen sayıların ondalık düzendeki gösterimini bulunuz.

11011011_2 10111011_2 00110101_2 11100010_2

1.8 Bilgisayarda bir mektup yazmak ve grafik çizmek isterseniz, bunu nasıl başarır siniz? *Uygulama programları deyimi size ne ifade ediyor*?

1.9 Çözmek istediğiniz herhangi bir problemi bilgisayara nasıl anlatırsınız? Bilgisayar size nasıl cevap verir?

1.10 Aşağıda ikilik düzende verilen sayıları oktal ve hekzadesimal sayı karşılıklarını bulunuz.
(a) 1110010110101101_2 (b) 1110111101_2 (c) 1001011100111111_2

1.11 Aşağıdaki sayıları onluk ve ikilik düzende temsil ediniz.

(a) 377_8 (b) $1A8_{16}$ (c) 111_8 (d) $1FF_{16}$

1.12 Aşağıdaki sayıları hekzadesimal düzende temsil ediniz.

(a) 57_{10} (b) -25_{10} (c) 1024_{10} (d) -1024_{10}

BÖLÜM 2
FORTRAN 90/95 DİLİNİN ESASLARI

2.1 BİR FORTRAN DEYİMİNİN YAPISI

Bir FORTRAN programında, programcının yapmak istediği işlemi gören bir dizi deyimlerin bir araya getirilmesinden ibarettir. Deyimler **yürütme** (icra) deyimleri (toplama, çıkarma, çarpma vs aritmetik ve mantıksal ifadeler ve bunlarla işlemler) ile **yürütme içermeyen** deyimlerden (programın bilgi deyimleri) oluşur.

İcra edilebilen bir Fortran satırı en fazla 132 karakter uzunluğa ulaşabilir. Bir Fortran deyimi tek bir satıra sığdırılamıyorsa, ve ("&") işareti kullanılarak bir alt satırdan devam edilebilir. Serbest-kaynak formunda (*free-source form*) yazılan Fortran deyimleri programın herhangi bir sütunundan başlayabilir. Örneğin,

```
Abc = a + b + c     ! üç değerin toplanması deyimi

Abc = a + b  &
    + c             ! iki satırda üç değerin toplanması

Abc = a + b &
    & + c           ! iki satırda üç değerin toplanması
```

Bir icra deyimi *en fazla* 40 satırdan oluşabilir.

Noktalı virgül (`;`) satır ayracı olarak kullanılmaktadır. Çok kısa deyimleri ayrı satırlarda yazmak yerine tek bir satıra sığdırmak amacıyla kullanılabilir. Örneğin,

Satır	**İfade**
1.	a=1.5
2.	x=3.*a-SIN(a)
3.	y=(a+x)/(x*x-3.*a)

program deyimlerini aşağıdaki gibi tek satırda girebiliriz:

```
a=1.5; x=3.*a-SIN(a); y=(a+x)/(x*x-3.*a)
```

Sabit-kaynak formunda (*fixed-source form*) her satır 80 sütundan oluşur. 1-5 sütunlar arası format ve satır etiketleri için kullanılır. Birinci ünlem ("`!`") karakterinin kullanılması satırın *açıklama* satırı olduğunu belirtir. Altıncı sütun normalde boş bırakılması gerekir ancak bir Fortran deyimi bir satıra sığmıyorsa 6.cı sütuna herhangi bir karakter konularak deyimin bu satırı da işgal ettiği belirtilmiş olur. 7 ile 72 sütunlar arası Fortran deyimlerinin yazılabileceği sütunlardır.



Fortran 77 sabit-kaynak formu kullanılır ve Fortran 90'da da kulanılabilmektedir. Ancak Fortran 90/95 programı hazırlarken serbest-kaynak formunu kullanmaya özen gösteriniz!

2.2 BİR FORTAN PROGRAMININ YAPISI

FORTRAN kaynak programları, öncelikle bir yazılımı kullanılarak hazırlanır ve ASCII olarak, yani yazıldığı gibi programcı tarafından okunabilen bir formatta, kaydedilir. Fortran kaynak programları belirli bir sırada yazılan yürütme ve yürütme içermeyen deyimlerin karışımından oluşur. Bir programı üç temel kısma ayırabiliriz: (1) *Tanımlama* kısmı, programın başında yer alır ve yürütme içermeyen deyimlerden oluşur: programda kullanılan değişkenler ve sabitlerin tipleri (tamsayı, gerçek, mantıksal, vs) tanımlanır; (2) *Yürütme* kısmı, program tarafından yapılması istenen işlemleri içeren bir veya daha fazla deyimden ve satırdan oluşur; (3) programın *çalışmasını durduran* deyim(ler), derleyiciye programı durdur komutunu verir.

2.2.1 TANIMLAMA KISMI

Tanımlama kısmı, ana programda, `PROGRAM` deyimi ve bunu takiben program ismini içerir. Alt programlarda `FUNCTION` veya `SUBROUTINE` ismini alır. Bu deyim her programın ilk satırıdır. Bu satırı tanımlama satır(lar)ı izler. Aşağıdaki örnek programda `ilk_program` programın ismidir. `INTEGER` satırında ise a, b ve c'nin tamsayı olduğu belirtilmiştir.

```
PROGRAM ilk_program

!  Programda kullanılan değişkenlerin tanımlanması kısmı
INTEGER :: a, b, c

!  Bir biri ile çarpılacak iki tamsayının programa okutulması
WRITE(*,*)'Çarpılacak olan A ve B sayılarını girin'
READ(*,*) a, b

!  İşlem kısmı : a ve b sayılarının çarpılması
c = a * b

!  Sonucun yazdırılması
WRITE(*,*)'Sonuç=', c

!  İşlem ve program sonu
END PROGRAM
```

2.2.2 YÜRÜTME KISMI

Programın kullandığı veriler ya ekrandan veya kütükten ya program içinde atama deyimleri ile ya da her iki yolla programa girdi olarak temin edilir. Program amacına uygun çeşitli işlemler yapıldıktan sonra sonuçlar ekrana, kütüğe veya hem ekran hemde kütüğe gönderilir. Bu amaçla `READ` ve `WRITE` gibi deyimler belirtilen işi gören (*okuma* ve *yazma*) deyimleri kullanılır. Aritmetik veya mantıksal işlemler yürütme deyimleri arasında yer alır. Örneğin, yukarıdaki örnekte `c=a+b` işlemi de bir yürütme deyimdir.

2.2.3 SONUÇ KISMI

Programın sonuç kısmı STOP ve END PROGRAM/FUNCTION/SUBROUTINE deyimlerinden birinden oluşur. END PROGRAM deyimine ulaşıldığında programın kendisi otomatik olarak STOP deyimini koyar ve programı durdurur. END FUNCTION/SUBROUTINE deyiminde ise bir RETURN deyimi ekler ve çağırıldığı programdaki satırın sonuna geri döner.

2.3 SABİTLER

Sabit, programda belirli bir sayıyı tanımlar. Tamsayı (INTEGER) ve gerçek sayı (REAL, DOUBLE PRECISION) sabitler olmak üzere iki tür sayısal sabit kullanılır. Sayısal sabitler, ayrıca, basamak sayısı itibarıyla sınıflandırılır. Bir sayısal sabit 0-9 basamaklarından oluşur. Rakamlar arasına boşluk bulunabilir. FORTRAN dilinde sayısal sabitlere ilave olarak, çeşitli amaçlar için, mantıksal (LOGICAL) sabitler (doğru=.TRUE. veya yanlış=.FALSE.), karmaşık sayı (COMPLEX), alfa sayısal sabitler (CHARACTER) de kullanılır.

2.3.1 INTEGER (TAMSAYI) SABİTLER

FORTRAN 90/95'de bir INTEGER (tamsayı) sabiti, ondalık noktasına sahip olmayan herhangi bir sayı olarak tanımlanır. Pozitif ve negatif değerler alabileceği gibi, "*sıfır*" değerini de alabilir. Tam sayının "pozitif" olduğu durumlarda artı işareti kullanılmayabilir. INTEGER olarak tanımlanan bir tamsayı sabitinin alabileceği değerler -2147483648 (-2^{31}-1) ile 2147483647 (2^{31}- 1) arasında değişir ve default (otomatik ayar) olarak her INTEGER sabit için 4 bayt hafıza gerektirir. Örnekler:

```
1234      1900     -567     +3289        12 345
 345_4    +89_tur1          B'10101'     Z"1AC3"
```

Sabitlerin arasında boşluklara müsaade edildiğinden, 12 345 geçerli bir tamsayı sabitidir ve karşılığı 12345 dir; 345_4 sabiti 4 baytlık tamsayıyı belirtmektedir. +89_tur1 sayısıda tip tanımlamada tur1 olarak tanımlanmış bir tamsayıdır. B'10101' ve Z"1AC3" geçerli tamsayı sabitlerdir. B (binary), O (oktal) ve Z (hegzadesimal) harflerini takiben tırnak içinde verilen sabitlerdir.

INTEGER*1 tanımında, hafızada bu tamsayı sabiti için 1 bayt yer ayrılır ve sabitin alabileceği tamsayı değerler aralığı -127 (-2^7 + 1) ile 127 (2^7 - 1) arasındadır. Diğer taraftan, INTEGER*2 tanımında, sabitin alabileceği tamsayı değerler aralığı -32767 (-2^{15} +1) ile 32767 (2^{15} - 1) arasında olup sabit için 2 bayt'lık hafıza yeri ayrılır.

Tamsayıların yanlış kullanımlarına bazı örnekler ise aşağıda verilmektedir.

```
1,234    -319.000  -255,670.1
```

Buradaki hatalar sayı içinde virgül, ondalık noktası veya her ikisinin kullanımından kaynaklanmaktadır.

2.3.2 REAL (GERÇEK SAYI) SABİTLER

Bir ondalık sayıya REAL (yani *Reel* veya *Gerçek*) sayı denir. Bu sayılar aynı şekilde pozitif veya negatif olabilirler. REAL sabitlerin alabileceği değer aralığı 1.18E-38 (1.18×10^{38}) ile 3.4E+39 (3.4×10^{38}) arasındadır ve kaynak program yazılırken, gerçek sabitlere mutlaka ondalık noktası konmalıdır. Bir REAL sabit için hafızada 4 bayt yer ayrılır.

REAL sabitler iki türlü ifade edilebilir: (i) *esas* REAL sabit ve (ii) eksponansiyel; yani *üstel*, şekilde ifade edilebilen REAL sabit. REAL sabitlerden, (i) grubuna örnekler şöyle verilebilir:

```
2.3187            4.                .2
0.00015      -1703.687003         1.24_4
```

Buradaki örneklerden 1.24_4 sayısının 4 bayt olarak tanımlanan 1.24 değerli bir sabit olduğunu belirtir.
Üstel gerçek sabit "E" (veya "e") harfi içeren ve on'un kuvveti ile çarpımını veren sayılardır. Bu sayı "*esas*" ve "*üst*" olarak iki kısımdan oluşur ve **aEn** ($=a\times10^n$) şeklinde belirtilir. Esas kısım "*a*" ondalık noktası içeren bir sayıdır. Üst kısmı ise "*n*" tamsayı olmalıdır!

REAL sabitlerin doğru kullanıma ilişkin bazı örnekler aşağıda verilmiştir:

2.1E2	210	→ 2.1×10^2
2.1e2	210	
.0004E+3	0.4	→ $.0004\times10^3$
.135e-2	0.00135	→ $.135\times10^{-2}$
1.2334E-1	0.12334	→ 1.2334×10^{-1}
1.23 34e-01	0.12334	

Yukarıdaki örnekleri incelediğimizde esas kısımlar sırasıyla 2.1, 0.0004 ve 0.135 tir; diğer taraftan üstler 2, +3 ve -2, yani tamsayı, olmaktadır.

REAL sabitlerin yanlış kullanımına bazı örnekler aşağıda verilmektedir:

```
32,129.5       344      -1.7873E-0.5         1/2
```

Yukarıdaki örneklerde yapılan hatalar şöyle açıklanabilir; (a) 32,129.5 sayısındaki virgül kullanımı, (b) 344 sayısının ondalık noktası içermemesi, (c) üstün 0.5 olması, yani tamsayı olmaması, (d) 1/2 sayısının kesirli olarak kullanımıdır (çünkü kesir en yakın tam sayıya yuvarlanarak 1/2=0 tamsayı değerini alacaktır).

2.3.3 DOUBLE PRECISION (ÇİFT HASSASİYETLİ) SABİTLER

Bir DOUBLE PRECISION (DP) sabit de aslında bir REAL sayıdır; yanlız DP sayının basamak hassasiyeti daha fazladır. Bu nedenle çift *hassasiyetli* sabitler olarak anılırlar. Bir DOUBLE PRECISION sabit 2.23D-308 (2.23×10^{-308}) ile 1.79D308 (1.79×10^{308}) arasında değerler alabilir. (*Not:* Burada "D" veya "d" harfi eksponansiyel DOUBLE PRECISION sabiti ifade etmektedir ve "E" ile tamamen aynı işleve sahiptir). Bu sabitler için hafızada 8 bayt yer ayrılır ki bu durumlarda gerçek sayının ondalık noktasından itibaren 15 ile 16 basamağa kadar kısmının bellekte saklanmasına olanak sağlar. Aritmetik işlemler DP

programlamada 15-16 basamak doğrulukla yapılır. Oysa bellekte saklama ve işlem yaparken bu basamak hassasiyeti REAL sabitlerde 7 ile 8 arasında değişmektedir. `REAL(KIND=8)` tip tanımı `DOUBLE PRECISION` tanımına karşılık gelir ve bellek gereksinimi her iki durumda da 8 bayttır. Örnekler:

Sabit	Karşılığı
l.0d+1	10.0
.1D0	0.1
13.156D-2	0.13156

2.3.4 `COMPLEX` (KARMAŞIK SAYILI) SABİTLER

Bir `COMPLEX` sabit ile, kompleks (karmaşık) sayı temsil edilir ve bu sayı bir çift `REAL` sayının belirli bir düzende ifade edilmesidir. Bu düzen `(1.,2.)` şeklinde olup, ilk sayı karmaşık sayının "gerçek" kısma ve ikinci sayı da "sanal" kısma karşılık gelir (yani, $1+2i$). Karmaşık sayı sabiti "`(`","`)`" parantezleri içinde yer alır ve gerçek ile sanal kısım birbirinden virgül ile ayrılır. Bir karmaşık sayı iki gerçek sayıdan oluştuğundan dolayı, bellekte saklanması için 4+4=8 bayt gerektirir. Bazı örnekler aşağıda verilmiştir:

```
(0.,0.)        (.0,-3.2)       (4.3e+2,-2.7E-3)
(1,3)          (1.e2,3.e-2)    (-3.d14,2.3_8)
```

Karmaşık sayılar çift hassasiyetli, yani `DOUBLE PRECISION`, olarak tanımlanmak istendiğinde tip tanımlama `COMPLEX*16` olarak yapılmalıdır çünkü karmaşık sayı iki çift hassasiyetli gerçek sayıdan oluştuğundan dolayı bellekte saklanması için 8+8=16 bayt gerekir.

2.3.5 `LOGICAL` (MANTIKSAL) SABİTLER

Bir `LOGICAL` (mantıksal) sabitin (`LOGICAL*l` veya `LOGICAL*4`) alabileceği değerler '*doğru*' (TRUE) veya '*yalnış*' (FALSE) tır. Atama esnasında TRUE veya FALSE ifadesinin önüne ve arkasına nokta konarak kullanılır.

Kullanım Şekli	Karşılığı
.TRUE.	*DOĞRU*
.FALSE.	*YANLIŞ*

Örnekler:

Doğru Kullanım	Yalnış Kullanım
A = .TRUE.	A = TRUE
OK = .FALSE.	OK = FALSE.

2.3.6 `CHARACTER` (ALFA SAYISAL, KARAKTERLERDEN OLUŞAN) SABİTLER

Bir `CHARACTER` sabiti, tek veya çift tırnak işareti (`'`) veya (`"`) ile kapatılan ve sembol ile harflerden oluşan alfa sayısallardır (*string*). Alfa sayısal, bellekte içerdiği boşluklar ile beraber ekranda gözüktüğü gibi, depolanır. Bir `CHARACTER` sabitinin içereceği minimum sembol sayısı *bir*'dir. Maksimum sayı ise kullanılan derleyiciye bağlıdır; ama genelde bu sınır 32767'dir. Alfa sayısal sabitin uzunluğu (karakter sayısı) iki tırnak işareti içinde kalan karakter sayısıdır. Bu sayı aynı zamanda alfa sayısalın hafızda tutulması için gerekli "bayt"

sayısını verir. Örneğin aşağıda verilen tabloda çeşitli alfa sayısallar ve uzunlukları verilmektedir (burada '_' ler boşluğu ifade ediyor).

Sabit	Değer	Uzunluk
`'abc'`	`abc`	3
`'ab"c'`	`ab"c`	4
`'a_bc'`	`a_bc`	4
`"a__bc"`	`a__bc`	5

Bazı geçerli `CHARACTER` sabitlerine örnekler:

```
'TEL: (222) 2345678'     'AHMET'        'NO ; '
'$1,200'                 'OY = %35'     '10,000 TL'
"CADDE : "               "x"
```

2.4 DEĞİŞKENLER

Normal cebirsel işlemlerde olduğu gibi, fiziksel miktarlar sembol veya harflerle ifade edilmekte (örneğin kütle m, yoğunluk ρ v.s) ve bunlarla $m = \rho V$ gibi aritmetiksel işlemler yapılabilmektedir. Formüllerdeki bu sembol veya harflerin her birine *değişken* denir. Ancak FORTRAN dilinde, programın başından sonuna kadar aynı amaçla kullanılan, bir veya daha fazla harf veya rakam karakterinden oluşan isimlere *değişken* adı verilir. Değişken isimleri aşağıdaki kurallara tabidir:

1. Değişken ismi harf veya sayıların kombinasyonu olabilir; fakat *mutlaka* İngilizce alfabesindeki bir harf ile başlamalıdır. Özel karakter veya sembollerin kullanımına (+, —, /,*, ...,α, β, Φ, Ω, π, &, #, ...) müsaade edilmez. Ayrıca Türkçe alfabesinde kullanılan ı, İ, ç, Ç, ş, Ş, ğ, Ğ, ü, Ü, ö, Ö harfleri de hiç bir suretle değişken veya sabit tanımında kullanılamazlar.
2. Bir değişken ismi en fazla 31 karakter ile sınırlıdır. Yalnız bazı derleyicilerde ANSI Standardının dışına çıkarak daha uzun isim tanımına müsaade etmektedirler. Bu nedenle piyasada mevcut çeşitli FORTRAN derleyicilerin kullanım kılavuzuna başvurunuz. Değişken isimlerinde under score adı verilen ("_") karakteri kullanılabilir.
3. FORTRAN dilinin `READ`, `PRINT`, `IF`, `DO`, `REAL` ve benzeri deyim ve komutlarının isimleri bire bir değişken ismi olarak kullanılamaz.
4. Aynı değişken ismi aynı program veya alt programda, farklı amaçlar için, birden fazla kullanılamaz. Ancak farklı alt programlarda aynı değişken isimi farklı amaçlarda kullanılabilir.
5. Programda kullanılmadan önce, bir değişken ismine, değişkenin tipine uygun bir değer atanmalıdır. Aksi takdirde aldığı ilk değer bellek değeri olacaktır. Değişkenin bellek değeri değişkenin tipine göre, tamsayı değişken için (`0`), gerçek değişken için (`0.0`) veya alfa sayısal değişken için "boşluk" (yani `' '`) olacaktır.
6. Bir değişken ismine `DIMENSION` deyimi ile boyut kazandırılarak, indisli değişken adını verdiğimiz değişken haline getirilebilir. Böylece indisli değişkenler ile bir isim altında çok sayıda değişkeni sayısal, mantıksal veya alfa sayısal değeri temsil etme olanağı sağlanır. Örneğin, $[a]=[a_1,a_2,a_3,\cdots,a_n]$ tek indisli, $[b]=[b_{1,1},b_{1,2},b_{1,3},\cdots,b_{1,n},\cdots$ $b_{2,1},b_{2,2},b_{2,3},\cdots,b_{2,n},\cdots]$ ise iki indisli değişkenleri temsil etmektedir.

Değişkenler, bilgisayar terminolojisinde bellek alanının adı olarak tanımlanır. Hatırlanması gereken önemli bir özelliği, bellek alanına bilgi yazıldığında önceki bilgilerin silineceği, bellek alanından bilgi okunduğunda ise bellek alanındaki bilgilerin aynen korunacağıdır.

Değişkenler ve sabitler, programın başında, tiplerine göre tanımlanmalıdır (`REAL`, `INTEGER`, `LOGICAL`, `COMPLEX` v.s.). Program başında tanımlanmayan değişkenler FORTRAN derleyicisinin otomatik (default) tanımına tabi olurlar. Otomatik tip tanımına göre, ilk harfi `I`, `J`, `K`, `L`, `M`, `N` olan harflerle başlayan değişken veya sabitler `INTEGER`, diğer harflerle başlayanların ise `REAL` olduğu kabul edilir.



Program yazarken kullandığımız sabit ve değişken isimleri, hem kendimizin hem de başkalarının takip edebilmesi bakımından, anlamlı, anlaşılır ve pratik olmalıdır.

2.4.1 SAYISAL VE MANTIKSAL DEĞİŞKEN TANIMLANMASI

Belirli bir tipteki değişkenler program başında belirtilmelidir. Belirtimin en genel şekli

```
<Tip> [,<nitelik listesi>] :: <değişken listesi> [=<değer>]
```

olarak verilmiştir. Burada "`::`" esasen opsiyoneldir; yani kullanılmasa da olur, ancak kullanmanın bir zararı yoktur. Ancak atama satırı `<nitelik listesi>` veya `=<değer>` içeriyorsa `::` kullanılması zorunludur. `<nitelik listesi>`, **`PARAMETER`**, **`SAVE`**, **`INTENT`**, **`POINTER`**, **`TARGET`** ve **`DIMENSION`** gibi niteliklerden oluşabilir. Bir değişken yukarıdaki bir veya daha fazla nitelik içerebilir. Aşağıda "geçerli" tip atamalarına örnekler verilmiştir:

```
REAL :: x, y, z
INTEGER :: i, j, sayi
LOGICAL, POINTER :: devre_1, devre_2, devre_3
REAL :: Bir_degisken_ismi_31_karakter
REAL, DIMENSION(10,10) :: y, z(10)
DOUBLE PRECISION, DIMENSION(0:9,0:9) :: cb
```

Bir satırda aynı tipteki birden fazla değişkene ilk veya başlangıç değerleri atanabilir. Bu işlem, tip tanımlamasını takiben değişken eşit işareti ile aşağıdaki gibi yapılır.

```
INTEGER   :: i=12, j=99
REAL      :: max_deger=10.e5
LOGICAL   :: acik=.TRUE., kapali=.false.
```

Bir tarihi (gün, ay, yıl) üç tamsayı olarak kullanmak istiyorsak, otomatik (default) tip tanımlama ile `gun`, `ay` ve `yil` isimlerini `INTEGER` olarak kullanmak mümkün değildir. Bunları aşağıdaki gibi `INTEGER` olarak tanımlamalıyız.

```
INTEGER :: gun, ay, yil
```

Bu tür tanımlamaların yapıldığı deyimlere belirtme veya *tanımlama* veya *bildirim* deyimleri (Specification Statement) denir. Bu listede yer alan değişkenler virgül ile ayrılan geçerli (yani, yukarıda bahsedilen değişken kurallarına uyan) FORTRAN isimleridir. Aynı zamanda, REAL olarak tanımlanan bir değişken INTEGER olarak (ya da tersi) bildirmi yapılamaz. Örnekler:

```
REAL    :: kutle, ivme, hiz
INTEGER :: top, sayi, tsayi
```

Bu örneğe dikkat edildiğinde, kutle ve ivme değişkenleri otomatik tip bildiriminde INTEGER bildirisi (K ve I harfleri) ile başladığı görülür. Oysa kütle, ivme ve hız bir tamsayı olamayacağı için, bu isimleri yukarıdaki gibi REAL olarak tanımlamamız gerekir.

2.4.2 KARAKTER VE ALFA SAYISAL DEĞİŞKEN TANIMLAMASI

Karakter veya alfa sayısal tipteki değişkenlerin tanımlanması, aynı sayısal değişkenlerin tanımlanmasında olduğu gibi yapılır. Ancak tip tanımlama için **CHARACTER** deyimi kullanılır. Belirtimin en genel şekli

```
CHARACTER [,(LEN=<karakter uzunluğu>)] [,<nitelik
          listesi>] :: <değişken listesi> [=<değer>]
```

Bazı örnekler

```
CHARACTER(LEN=12) :: isim, soyad
CHARACTER         :: cinsiyet
CHARACTER(LEN=32) :: cadde, sokak*25
CHARACTER         :: AY*7,  YIL*4,  GUN*8
```

Yukarıdaki örneklerde isim, soyad için 12, cinsiyet için 1, cadde için 32 ve sokak için 25 karakter uzunluk (veya bayt) atama yapılmıştır. Programda kullanılan alfa sayısal değişkenlerin uzunlukları aynı ise, örneğin, Ad, Soyad ve Adres maksimum 20 karakter uzunlukta olması isteniyorsa,

```
CHARACTER(LEN=20) :: Ad, Soyad, Adres
```

şeklinde de tanımlanabilir. Diğer bir tanımlama

```
CHARACTER(LEN=12) :: Ad, Soyad, Tel*9, Adres*55
```

şeklinde olabilmektedir. Bu tanımlama ile Ad ve Soyad 12, Tel 9, Adres ise 55 karakter uzunluğunda olacağı bildirilmektedir.

2.4.3 SEMBOLİK SABİTLERİN TANIMLANMASI

Bir program içinde değişken tanımlama satırlarında kullanılmak üzere sembolik sabitler oluşturulabilir. **PARAMETER** ile tanımlanan sabitler, tanımlama kısmında kullanılabilir. Genel kullanım şekli aşağıda verilmiştir.

```
<Tip>, PARAMETER :: <değişken_1=değer,... >
```

Örnekler:

```
REAL, PARAMETER :: pi=3.14159
INTEGER, PARAMETER :: nsatir=10, nsutun=12
REAL, DIMENSION(nsatir,nsutun) :: matrisA, matrisB
CHARACTER(LEN=5), PARAMETER :: p1="Parti1",p2="Parti2"
```

2.4.4 SABİT VE DEĞİŞKENLERİ KIND İLE TANIMLAMA

FORTRAN'da değişken veya veri türü belirleme işlemi, tek veya çift hassasiyet türü, **KIND** ile yapılabilir. Tek ve çift hassasiyet bildiriminde her türün kendine has bir KIND-sayısı vardır. Örneğin,

```
REAL(KIND=1) :: deger1
REAL(KIND=4) :: deger2
REAL(KIND=8), DIMENSION(20) :: vektor
REAL(4) :: deniz
```

REAL değişkenin türü parantez içinde **KIND=** ifadesi kullanılarak veya kullanmadan belirtilir. Eğer KIND kullanılmaz ise tür için otomatik ayar değeri kullanılır bu değer 4'dür.

Çeşitli Fortran derleyicilerinde tür deyimi farklı sayı türlerini ifade edebilir. Örneğin, 32-bit'lik bir gerçek sayı bir derleyicide KIND=1, 64-bit'lik bir gerçek sayı da KIND=4 olabilirken, başka bir derleyicide, 32-bit'lik bir gerçek sayı KIND=4, 64-bit'lik bir gerçek sayı da KIND=8 olabilmektedir. Bu durumu açıklığa kavuşturmak için, kullandığınız derleyicinin El Kitabına başvurunuz. Bu nedenle, farklı derleyicilerle çalıştırabilmek için programda en az değişikliği yapmanızı gerektirecek şekilde programınızı yazınız.

```
INTEGER, PARAMETER :: tek= 4   ! Derleyiciye bağlı bir değer
INTEGER, PARAMETER :: cift=8   ! Derleyiciye bağlı bir değer
REAL(KIND=tek) :: deger2
REAL(KIND=cift), DIMENSION(20) :: vektor
REAL(tek) :: deniz
```

Derleyici gerektirdiği için yukarıdaki programda tek hassasiyet için KIND=1 ve çift hassasiyet için KIND=4 tanımlaması yapılması gerekirse, programın sadece iki satırında değişiklik yapılması gerekecektir. REAL sabitleri tanımlarken de türünü belirtmek mümkündür:

```
34.15        ! Tek hassasiyet(KIND=4)
34.15_4      ! KIND=4
34.15E3      ! Tek hassasiyet(KIND=4)
34.15_cift   ! cift PARAMETER deyimi ile 8 b olarak atanmış olmalı
34.15D3      ! Çift hassasiyetli
```

2.4.5 HASSASİYETİN İŞLEMCİDEN BAĞIMSIZ OLARAK BELİRLENMESİ

Bir FORTRAN programını kullanmada karşılaşılan problemlerden birisi de, derlenmiş programı farklı bilgisayarlarda çalıştırmaya kalktığınızda "*tek*" veya "*çift*" hassasiyetin bit tanımlamasının kaynaklanan farklılıkların oluşabilmesidir. FORTRAN 90/95 ile kullanılan sayıların basamak hassasiyeti ve üst sınırlarını program içinde tanımlamak mümkün olmaktadır; böylece bir işlemciden bir başkasına tip farklılığından doğan sorun giderilir. Bu işlem aşağıdaki şekilde gerçekleştirilir:

```
Kind_Nosu=SELECTED_REAL_KIND(p=hassasiyet,r=aralık)
```

Burada **p** noktadan sonra arzu edilen basamak hassasiyeti, **r** de 10^r olarak algılanan, sayının üslü sınırıdır. Bu iki değerden sadece birisi de kullanılabilir; yani her iki seçeneği kullanmak zorunluluğu yoktur. Aşağıda bir dizi geçerli tanımlama deyimleri verilmiştir:

```
Kind_Nosu = SELECTED_REAL_KIND(p=6,r=37)
Kind_Nosu = SELECTED_REAL_KIND(p=12)
Kind_Nosu = SELECTED_REAL_KIND(r=100)
Kind_Nosu = SELECTED_REAL_KIND(13,200) ! p=13, r=200
Kind_Nosu = SELECTED_REAL_KIND(13)     ! p=13
Kind_Nosu = SELECTED_REAL_KIND(p=17)
```

Örnekler

```
INTEGER, PARAMETER:: tek =SELECTED_REAL_KIND(p=6,r=37)
INTEGER, PARAMETER:: cift=SELECTED_REAL_KIND(p=13,r=200)
REAL(KIND=tek) :: Sayi_1, Sayi_2, Toplam=0.0
REAL(KIND=cift):: A, B, C, Sayi=0.0_cift
```

şeklinde kullanılabilir.

2.5 ARİTMETİK İFADELER

Aritmetik ifadeler daha önce değer atanan sabit veya değişkenlere birden fazla aritmetik işlemin uygulandığı talimatlardır. İfade `C=A+B` gibi oldukça basit olabileceği gibi, birkaç satırı işgal edebilecek uzunlukta ve karmaşıklıkta da karşımıza çıkabilir. Aritmetik işlemler yapılarak, ifadenin solundaki değişkene hesaplanan bir tek değer atanır.

Normal aritmetik işlemler FORTRAN'da aşağıdaki semboller ile gerçekleştirilir.

+	Toplama	*	Çarpma
–	Çıkarma	/	Bölme
**	Üst alma		

Aritmetik ifade oluşturmak için ortaya konan kurallar şunlardır:

1. Bir işlemde iki aritmetik operatör yan yana, `+-`, `++`, `*/` (`**` *hariç*) gibi, kullanılamaz.

2. Çarpma, bölme ve üst alma işlemleri iki sayı veya değişken ile beraber kullanılır. Örneğin,

```
X*Y          Z2/3.0          A**4
```

FORTRAN'da aritmetik ifadelerde dikkat edilecek hususlar şunlardır:

1. Çarpma işlemi cebirdeki şekilde basitleştirilemez. Örneğin, *a(b+c)* ifadesi **`A(B+C)`** olarak değil **`A*(B+C)`** olarak yazılmalıdır.
2. Yan yana kullanılan operatörlere *dikkat* edilmelidir. Örneğin,

Geçersiz ifade	*Geçerli ifade*	*Cebirsel ifade*
`X**-3`	`X**(-3)`	x^{-3}
`X-+D`	`--`	Belirsiz
`X++D`	`--`	Belirsiz
`X*/D`	`--`	Belirsiz
`B*-3.0`	`B*(-3.)` veya `-3.*B`	*-3b*

3. Bazı aritmetiksel terimler, cebirde olduğu gibi, parantezler ile gruplandırılabilir. Ancak *sadece* '`(`', '`)`' parantezleri kullanılabilir; { } veya [] parantezleri FORTRAN 90/95'de kullanılmaz, ancak FORTRAN 2003'de farklı amaçlarda kullanılır.

2.5.1 İŞLEM SIRASI

Bilgisayarın, aritmetik işlemleri yaparken, izlediği bir dizi kural vardır. Bunlar:

1. Parantez içindeki işlemlerin hesabı öncelikle yapılır. En içteki paranteze alınmış hesaplar önce, dıştakilerin hesabı en son yapılır.
2. Üstel ifadeler (**) diğer aritmetik işlemlerden önce hesaplanır.
3. Çarpma ve bölme işlemleri eşit önceliğe sahiptir.
4. Toplama ve çıkarma (eşit öncelikli) işlemleri yapılır.
5. Aynı derecede önceliğe sahip aritmetik işlemler (toplama ve çıkarma işlemlerinde olduğu gibi), soldan sağa doğru bir sırayı takip ederek hesaplanır.

Bu örnekleri incelersek `A+B+C` ifadesinde, önce `A+B` hesaplanır; sonuç `C` ile toplanır; diğer taraftan sadece `A**B**C` ifadesinin hesabında 5. kural ihlal edilmektedir: `A**(B**C)` ifadesinin hesabına eşdeğerdir; yani önce `B**C` daha sonra `A**(ust)`işlemi yapılır.

Örneğin, $a \cdot b + \frac{c}{d} f^2$ aritmetik ifadesinde işlem sırası (**1**) `F**2`, (**2**) `A*B`, (**3**) `C/D`, (**4**) `(3)*(1)`, (**5**) `(2)+(4)` şeklinde uygulanır.

Bazı aritmetik ifadelerin kullanımlarına ilişkin örnekler Tablo 2.l'de verilmiştir. Bunları dikkatle inceleyerek FORTRAN'da aritmetik işlemlerin uygulanmasını çok iyi bir şekilde kavramanız çok önemlidir.

Tablo 2.1: Aritmetik işlemlerin uygulanma şekli ve sıkça yapılan hatalar.

İşlem	*Yanlış İfade*	*Doğru İfade*
$x(y + z)$	`X(Y+Z)`	`X*(Y+Z)`
$3x + 2y$	`3*X+2*Y`	`3.*X+2.*Y`
x^{y+2}	`X**Y+2`	`X**(Y+2)`
$x^{1/y}$	`X**(1/Y)`	`X**(1./Y)`
$\frac{xy}{zt}$	`X*Y/Z*T`	`X*Y/(Z*T)` veya `X*Y/Z/T`
$\frac{a}{bc}$	`A/B*C`	`A/(B*C)`
$(-x)^y$	`-X**Y`	`(-X)**Y`
$\left(\frac{1}{5}\right)^x$	`(1/5)**X`	`(l./5.)**X` veya `0.2**X`
$x = 3.2 \times 10^6$	`X=3.2*10.**6`	`X=3.2E+6`

Aritmetik atama deyimlerini FORTRAN'ın algılamasına bazı örnekler aşağıda verilmiştir:

`x=a+b/5.-c**d+l*e` $\Rightarrow$ $x = a + \frac{b}{5} - c^d - l \cdot e$

`y=-a*b/d-c/d**e/f+g**h+1-j/k` $\Rightarrow$ $y = -\frac{ab}{d} - \frac{c}{d^e} + g^h + 1 - \frac{j}{k}$

`z=a+b/c+d-e/f*h` $\Rightarrow$ $z = a + \frac{b}{c} + d - \frac{e}{f}h$

Tamsayılar ile yapılan aritmetik işlemlerin sonucu da bir tamsayı ile sonuçlanır. Bu noktayı özellikle tamsayılarla bölme işlemlerinde *mutlaka* hatırlamalısınız; çünkü iki tamsayının bölümü bir tamsayı değilse, bölümün ondalık kesri atılarak tam kısımdan oluşan bir tamsayıya yuvarlanacaktır.

```
4/2=2,    8/2=4,   3/2=l,  4/3=l,   3/4=0
```

Bu nedenle tamsayı değişkenler, aritmetik işlemlerde genellikle, sayaç veya indis olarak kullanılırlar.

REAL sayılar ve değişkenler ile yapılan aritmetik işlemlerin sonucu da gerçek sayıdır ve tamsayılarda karşılaşılan problem burada oluşmaz.

```
4./2.=2.           8./2.=4.         3./2.=l.5
4./3.=l.333333     3./4.=0.75
```

Fakat burada bir başka, örneğin `l./3.=0.33333333` işleminde karşılaşılan tekrarlanan `3`'ler gibi, problem ortaya çıkar. Diğer bir deyişle, bilgisayar aritmetiğinde elde edilen sonuç tam olarak `1/3`'e eşit olmaz; yani

```
3.*(1./3.)-1.≠ 0.0
```

Bu problem genellikle iyi bir programcı için sorun yaratmaz; fakat acemi programcıların program yazarken, bu hususu akılda tutması ve programda REAL *sayılarla karşılaştırma yapmaktan kaçınması* tavsiye edilir.

Bir diğer örnekte: `4*3/2→12/2→6` ve `3/2*4→1*4→4` elde edilir (sayılar tamsayı olduğundan dolayı). Sonuç olarak `(N*2)/2` ifadesi aritmetiksel olarak `(N/2)*2` ifadesi ile eşdeğer olmasına rağmen FORTRAN açısından eşdeğer değildir. Bu yüzden buna benzer ifadelerle işlem ve program yaparken daha dikkatli olunmalıdır.



TAMSAYI(lar) ile yapılan aritmetik işlemlerde dikkatli olunuz! Arzuladığınızdan farklı sonuçlar elde edebilirsiniz.

x^a gibi üs alma işleminde "*esas*" (x) ve "*üst*" (a) sayılara dikkat edilmelidir, "*esas*" sayı gerçek ve "üst" tamsayı ise hiç bir problem yoktur. Yanlız esas ve üst negatif gerçek sayı ise, bu işlem logaritma bazında hesaplanmaya çalışıldığından (negatif bir sayının logaritmasını almaya çalışacaktır) sonuç vermeyecektir. Bu nedenle *esas* kısmın negatif olmamasına dikkat ediniz. Örneğin;

```
 (-3.)**3        ⇒   -27.
(-3.)**(-3)      ⇒   -1./-27.
(-3.)**(-3.)     ⇒   sonuçsuz
```



“üst” gerçek ise “esas” pozitif olmalıdır! Mümkün olduğunca üst değerinini TAMSAYI olarak ifade ediniz.

2.5.2 KARIŞIK MODDA ARİTMETİK İŞLEMLER

Bir aritmetik ifadede kullanılan değişken ve sabitlerin aynı tipte olmaları veya olmamaları sonucu etkileyecektir. INTEGER, REAL ve/veya COMPLEX sayıların karışık olarak kullanıldığı işlemler gerekebilir. Bu durumda, $C = A◊B$ (burada ◊, A ile B arasında dört işlemden herhangi birini tanımlıyor) işleminin sonucunu aşağıdaki tablo ile verilen kurallar belirler.

A◊B	**Tamsayı**	**Gerçek**	**Karmaşık**
Tamsayı	*Tamsayı*	*Gerçek*	*Karmaşık*
Gerçek	*Gerçek*	*Gerçek*	*Karmaşık*
Karmaşık	*Karmaşık*	*Karmaşık*	*Karmaşık*

Bazı FORTRAN derleyicileri karışık modda işlemlere müsaade etmezler. Bu bakımdan bir ifadede karışık tipte sabit ve değişkenlerin kullanımından kaçınılmalıdır.

YANLIŞ	*DOĞRU*
`2*A+3*B`	`2.*A+3.*B`
`2.*I+3.*J`	`2*I+3*J`
`B*B-4*A*C`	`B*B-4.*A*C`

REAL sayının INTEGER kuvvetinin hesabı karışık modda bir işlem olarak sayılmaz.

KARIŞIK İŞLEM	*SONUÇ*
3.+1/4	3. (**1/4→ 0**)
3+1./4	3.25 (**1./4→0.25, 3+0.25→3.25**)
3/2+3./2	2.5 (**3/2→ 1, 3./2→1.5, 1+1.5→2.5**)

Eğer A=B deyiminde, A'nın tipi B'nin tipinden farklı ise, B'nin sonucu A'nın tipine dönüştürülür. Gerçek sayıdan tamsayıya dönüştürmede, ondalık noktasından sonraki kısım göz ardı edilir.

```
INTEGER :: K
REAL    :: A, B
A = l.234
K = A + 1
PRINT*, K
B = K + 3
PRINT*, B
```

Programında K=2 ve B=5.0 değerlerini alırlar.

Programcılar karışık modda çalışma kolaylığının getirdiği külfetlerin daima farkında olmalıdırlar!: her tamsayı bir gerçek sayıya dönüştürülürken makinenin mikro saniye mertebesinde olsa bile belirli bir zamanı kullanılır. Böylece milyonlarca işlemin yapıldığı bir blokta yer aldığında, karışık modda çalışma, çalıştırma süresini artırır ve programın verimliliğini düşürme tehlikesi oluşur. Buna ilaveten, bu şekilde yazılan bir program başka derleyicilerde derlenmesinde problemler çıkarması olasılığı nedeniyle, program *daha az esnek* olacaktır.



Aritmetik işlemlerde, gerçek sayının basamak hassasiyetinin sınırlı olduğunu unutmayınız; bunların kullanımında çok dikkatli olunuz! Gerçek sabitler, tam değerlerden oluşsa bile, MUTLAKA ondalık noktasını koyunuz.

2.6 ARİTMETİK ATAMA DEYİMİ

Bilgisayarın bir aritmetik ifadeyi hesaplayabilmesi için bu ifadede yer alan bütün değişkenlere, değişkenler kullanılmadan önce, atama yapılması gerekir. Birçok derleyici tanımlanmayan değişken için *hata mesajı* verir.

Bir değişkene nasıl atama yapılır? Yani bellekteki değeri nedir? Bu soruların cevabı oldukça basittir. Bir değişkenin değeri ya programda A=2.3 gibi atama yapılarak ya ekrandan ya da kütükten okutma işlemiyle tespit edilir. Atama yapılmayan değişkenin değeri sayısal değişkenler için sıfır, alfa sayısallar için boş'tur.

M değişkeninin bellek alanına 18 sayısını yerleştirme işlemi, akım şemalarında veya algoritma oluşturma esnasında, sembolik olarak M ←18 ile gösterilir. Bu işleme *atama deyimi* denir. FORTRAN'da ← işareti, eşitlik ('=') işareti ile temsil edilir; dolayısıyla, M←18 atama deyimi M=18 olarak yazılır. Böylece M için ayrılmış olan bellek sahası 18 rakamı ile doldurulur. Diğer taraftan, bilinen bir x değişkeni için x'in karesinin (x^2) hesaplayıp, bu

değeri *Y* bellek sahasında depolamak istersek, bu atama işlemi sembolik olarak $Y \leftarrow x^2$ ile gösterilir ve programda `Y=X**2` şeklinde kodlanır.

Genel olarak bir FORTRAN *aritmetik atama deyiminin* kullanım şekli

`Değişken ismi=aritmetik ifade`

şeklinde olmalıdır.



Eşitliğin sol tarafı sadece bir tek değişken içermelidir. Sağ tarafında cebirsel ve/veya mantıksal ifadelerden oluşan işlemler yer alabilir. '=' ile '==' sembollerinin işlevlerini karıştırmayınız.

FORTRAN da atama teriminin "eşitlik" (=) sembolü ile temsil edilmesi; diğer bir deyişle, "=" sembolünün *eşitliği* ifade etmesi bakımından, bir talihsizliktir. Ancak bilgisayar programlarının tamamında, eşitlik sembolü tamamen farklı işlevde kullanılmaktadır. Atama terimi ile kastedilen, eşitliğin sağ tarafında verilen ifade ile hesaplanan değerin, sol taraftaki değişkenin bellek alanına yerleştirilmesi veya kopyalanmasından ibarettir. İleride değinileceği üzere, '==' sembolü mantıksal önermelerde eşitlik anlamında kullanılır.

Unutulmaması gereken nokta kısaca *(1) eşit sembolünün sağındaki aritmetik işlemler yapılır (2) bulunan değer eşitliğin sol tarafındaki değişkenin bellek alanına yazılır.*

2.7 BELLEK ALANININ KULLANIMI

Programda kullanılan her değişken için bir bellek alanı tahsis edildiğinden bahsetmiştik. İndisli değişkenlerde ise bellek alanı alt ve üst indisler ile belirlenir. Bu bellek alanlarını sonsuz uzunlukta bir film şeridine benzetirsek, her şeride sadece bir sabit veya değişkenin (sayısal, mantıksal, karmaşık veya alfa sayısal) değerinin yazılabilir. Program değişkenlerinin tipleri atandıktan sonra aldıkları başlangıç değerleri "sıfır" veya "boş" tur. Program içinde yapılan atamalar ile bu değerler güncellenir.

Bir bellek alanında sadece bir değer yazılabilir.. Bu bellek alanına yeni bir atama yapıldığında eski değer silinir; yerine yenisi yazılır. Eski ve yeni değerleri aynı anda bellekte tutmaz. Örneğin,

```
Satır No
   1.       REAL :: A, B, C
   2.       B=A**2+4.
   3.       C=A+B
   4.       C=SQRT(C)
   5.       PRINT*,  A,B,C
```

olarak verilen programda kullanılan değişkenler `A`, `B`, `C` dir. Dolayısıyla programın başında bellekteki değerleri (yani 1. satırdan önce), bu değişkenlere başlangıç değerleri atanmadığı için, `0.0` dır. İkinci satırda A$\leftarrow$0.0 olduğundan B$\leftarrow$4.0 elde edilir. Bu andan itibaren bellekteki alan değerlerinden sadece B'nin değeri değişmiştir, üçüncü satıra gelindiğinde halen A'nın değeri sıfır olduğundan C$\leftarrow$0.0+(4.0) ifadesinden C$\leftarrow$4.0 elde edilir; yani artık C'nin bellek değeri de 4.0 olmuştur. Dördüncü satıra geldiğimizde eşitliğin sağ tarafı C'nin bellek değerini kullanırken, sağ tarafta hesaplanan değer C'nin yeni bellek değerini

oluşturmaktadır. Bu bilgilerin ışığı altında C $\leftarrow \sqrt{4}$ =2 elde edilir ki C'nin yeni bellek değeri artık 2.0 dir.

2.8 GİRDİ/ÇIKTI LİSTESİ

Bir FORTRAN programda **READ*,** ve **PRINT*,** deyimleri sırasıyla ekrandan veri okuma ve çıktıyı ekrana yazdırma amaçları için kullanılır. Kısa ve basit programlar yazmaya başlayabilmek için, kısaca bu deyimlerin kullanımlarına değinelim. **PRINT*,** deyimi genel olarak

```
PRINT*, <değişken listesi, aritmetik ifade veya
          alfa sayısallar>
```

şeklinde kullanılır. Yukarıdaki şekilde kullanılan yazdırma deyimi ile konsola (yani ekrana) yazdırma işlemi yapılır. Burada * serbest formatı temsil eden bir karakterdir. Değişken listesindeki her bir değişken virgül ile ayrılır.

```
PRINT*, A, B, C
```

deyimi A, B ve C gerçek değişkenlerinin değerini ekrana yazar. Formatlı çıktı yazdırma deyimlerine ileride değinilmektedir.

```
PRINT*, 'X=',X, ' Y=',Y
```

deyimi de 'X=' ve ' Y=' ile verilen aslında alfa sayısal sabitleri X ve Y'nin değeriyle beraber, deyimde kullanılan sırayla ekrana yazılmasını sağlar. Örneğin X=3 ve Y=2 olsun. Bu durumda çıktı

```
X=3.000000 Y=2.000000
```

şeklinde görülür.

PRINT*, deyiminde (ve bunun gibi diğer çıktı deyimlerinde) aritmetik ifadelere de müsaade edilir; örneğin

```
PRINT*, A, B, A+B
```

deyiminde A ve B'nin değerlerinin yanında bunların toplamının ekrana yazılması sağlanır.

READ*, deyimi ile ekrandan girilmesi arzulanan değişken listesi eşlik eder.

```
READ*,   < değişken listesi >
```

Bu listeye karşılık girilen değerler, değişkenlerin tip tanımlarına (REAL, INTEGER vs) uygun olmalıdır. Aynı PRINT*, deyiminde olduğu gibi her bir değişken virgül ile birbirinden ayrılmalıdır. Örneğin,

```
REAL    :: A, X2, Z
INTEGER :: Ki, J48
READ*,  A, X2, Ki, Z, J48
```

deyimine girilen

```
2.35,   7.12,  718,   0.45,   6.17
```

değerleri

```
A=2.35    X2=7.12    KI=718   Z=0.45    J48=6
```

atamasına eşdeğerdir; örneğin `J48` bir *tamsayı* olarak tanımlandığından bellekte sayısal değerin sadece tam kısmı tutulur.

Çok sayıda değerin girilmesi gereken durumlarda ekrana hangi değerin girilmesi istendiğini belirten bir mesaj yazılması, programa yanlış veri teminini önler. Örneğin,

```
PRINT*, ' İkinci dereceden denklemin katsayılarını'
PRINT*, ' a, b ve c sırasıyla giriniz '
READ*, A, B, C
```

deyimleri ile kullanıcıya hangi verilerin, hangi sırayla girilmesi gerektiğini bildirir.



Ekrandan veri girişlerinde, kullanıcıya hangi verilerin hangi sırayla girilmesi gerektiğini **`PRINT*,`** *deyimi ile bildiriniz.*

2.9 IMPLICIT/IMPLICIT NONE DEYİMİ

Bu deyim, bir program içindeki değişkenlerin tiplerini değişken isim bazında değil de, değişken isminin ilk harfi bazında tanımlanmasında kullanılır. `IMPLICIT` deyimi ana veya alt programın (`FUNCTION` veya `SUBROUTINE` deyiminden sonra) ilk deyimi olmalıdır. Örneğin, `IMPLICIT INTEGER(A-Z)` deyiminde, alfabenin tüm harfleri ile başlayan değişkenlerin `INTEGER` tipinde olduğunu bildirir. `IMPLICIT REAL(A-H,O-Z)` deyimi ise başlangıç harfi A ile H ve arası O ile Z ve arasındaki harfler ile başlayan değişkenlerin `REAL` olduğunu bildirmektedir. Tip bildirimi yapılmayan harfler (yani `I, J, K, L, M` ve `N`), FORTRAN'ın *otomatik* bildiriminden dolayı `INTEGER` olarak kalırlar. Örneğin,

```
IMPLICIT INTEGER(A,C,H-K)  !  A,C,H-K arası ile başlayan değişkenler "tamsayı"
IMPLICIT REAL(P,R,S-W,Z)   !  P,R,Z, S-W arası ile başlayan değişkenler "gerçek"
IMPLICIT COMPLEX(A-C,Z)    !  Z ve A-C arası ile başlayan değişkenler "karmaşık"
IMPLICIT INTEGER(D-R)      !  D ile R arası ile başlayan değişkenler "tamsayı"
IMPLICIT LOGICAL(V,W)      !  V,W ile başlayan değişkenler "mantıksal"
IMPLICIT REAL*8(A-D,O-W)   !  A ile D ve O ile W arası ile başlayan değişkenler
                           !  "çift hassasiyetli" sayı
IMPLICIT COMPLEX*16(E-J,Z)!  E ile J arası ve Z ile başlayan değişkenler "çift
                           !  hassasiyetli karmaşık" sayı
```

FORTRAN dilinin otomatik değişken tip tanımlama (default) kuralı

```
IMPLICIT REAL(A-H,O-Z) ! A-H ve O-Z arası ile başlayan değişkenler
                         "gerçek", I-N ile başlayan değişkenler "tamsayı"
```

şeklindedir. Otomatik tip belirleme kuralını kapatmak için **IMPLICIT NONE** deyimi kullanılır. Böylece derleyici tip tanımlama satırında belirtilmeyen değişkenlerin tümü için "*değişken tanımlanmamış*" hata mesajı verir.



Programda kullandığınız tüm değişkenleri "tanımlama" satır(lar)ında belirtiniz ve **IMPLICIT NONE** *deyimini kullanınız ki derleyici eksik tanımlamaları ve yazım hatalarını bulap size bildirsin.*

2.10 PARAMETER NİTELİĞİ

Program tanımlama kısmında kullanılan bir sabite isim verme olanağı sağlayan bir bildiri deyimidir. Genel kullanım şekli,

```
<Tip>, PARAMETER :: p_ismi1 = p1, p_ismi2 = p2, ...
```

Bu şekilde tanımlanan isme *parametre* veya bir sabitin *sembolik ismi* denir. Sembolik isim tanımlandıktan sonra, programdaki bir ifadede sabit olarak kullanılır; yani bu isimin bellek değeri program boyunca *değiştirilmez*! PARAMETER deyimi içinde aritmetik deyim yer alabilir; ancak kütüphane fonksiyonları kullanılamaz.

Bir PARAMETER deyimindeki bir sembolik isim sadece o programdaki sabiti tanımlar. Sembolik sabitin tipi sembolik ismin tipine bağlıdır. Eğer sembolik isim bir CHARACTER sabitini belirtiyorsa, sabit tek veya çift tırnak işaretleri arasında soldan hizalanarak yazılmalı, tanımlanan uzunlukta olması için gerektiğinde boşluklar kullanılmalıdır.

Örnekler:

```
REAL, PAREMETER :: I=1
COMPLEX, PARAMETER :: C=I+1
. . .
PRINT*,   I, C
```

programının çıktısı

```
      1.0000000   (2.000000, 0.000000)
```

şeklindedir.

```
CHARACTER(LEN=8), PARAMETER :: U='l 2345 6',K='l 234  '
. . .
PRINT*,  U
PRINT*,  K
```

programının çıktısı

```
     1 2345 6
       1 234
```

olur; çünkü U aynı K 8 karakter uzunlukta tanımlanmıştır.

PARAMETER ile tanımlanan sabitler, tip tanımlama kısmında, aşağıdaki örnekte olduğu gibi, değişken gibi kullanılmasına olanak sağlar.

```
INTEGER, PARAMETER :: N=12, M=2*N
CHARACTER(LEN=N) :: AD
CHARACTER(LEN=M) :: SOYAD
```

örneğinde AD 12, SOYAD 24 karakter uzunluğunda tanımlanmaktadır.

Hem tanımlama hem de sabit atama bir arada yapılabilir.

```
INTEGER, PARAMETER :: N=15, MAX_sayi=9999
REAL, PARAMETER    :: epsilon=1.0e-6
LOGICAL, PARAMETER :: TUS=.FALSE.
CHARACTER(LEN=5), PARAMETER :: cevap='HAYIR'
```

ALIŞTIRMALAR

2.1 Bilgisayarınızın sayısal değerlerinin sınırı nedir? $[(\sqrt{2.})^2 - 2.]$ ifadesi ile $[(\sqrt{4.})^2 - 4.]$ ifadesini bilgisayarda hesaplayarak sonuçları karşılaştırınız ve yorumlayınız.

2.2 `X=l./3.+l./3.+l./3.` ile `Y=l./4.+l./4.+l./4.+l./4.` ifadelerini hesaplayan bir program yazınız ve sonuçların nedenlerini açıklayınız.

2.3 Aşağıdaki atama ifadelerindeki hataları bulunuz.

```
a) 2X=A              b) X+3*Y=Z          c) A2Z=2.35E-2
d) Y=l.403E-3.5*X    e) D=l              f) Ab=Y***X
```

2.4 Aşağıdakilerden hangileri geçerli değişken isimleridir.

```
Al            1A            HQXYŞ        BiR
X21Y9         3A000bx       asdB6        X-19
BC45          AB 12         235X         ALFA
B23.4         Bir:2         KEDİ          mil
KiLOMeTRe     J99X          XKIJ         34MN
N$12          kilometre     Kedi         na_34
yzk           kisi%adi
```

2.5 Aşağıdakilerden hangisi geçerli aritmetik deyimlerdir.

```
(a) X=A*C2                 (b) VOLT=AKIM*DIRENC
(c) ABCDEFGH=A-B           (d) U+V=UV
(e) 123=JKL                (f) kdv=0.15*fiyat
(g) yog=xm/V               (h) NNkk= *Nk2
(ı) gu=CC*3.14159          (i) Isq=K** I
(j) –AB=3.0                (k) X=x+x*X
```

2.6 Aşağıdaki FORTRAN deyimlerinde aritmetik işlem sırasını göz önünde bulundurarak sonucu bulunuz. (`a=3. b=2. c=5. d=4. e=10. f=2. g=3.`)

```
(a) cikti=a*b+c*d+e/f**g
(b) cikti=a*(b+c)*d+(e/f)**g
(c) cikti=a*(b+c)*(d+e)/f**g
(d) cikti=(a+b)*f**2/(c+d)/g**3
(e) cikti=(a*b+c*d)/f**g
(f) cikti=a**(b**c)
(h) cikti=(a**b)**c
(h) cikti=a**b**c
```

2.7 Aşağıdaki aritmetik ifadeleri FORTRAN dilinde kodlayınız.

(a) $\dfrac{5(a/c)^{m-1}}{(r-t)^{1/m}+(r+t)^{3/5}}$

(b) $3\times10^{8}\left(\dfrac{a+b}{b+c}\right)-7\times10^{-7}\left(\dfrac{a^2+b^2}{b^2+c^2}\right)$

(c) $\left[\dfrac{2(p/q)^{k-1}}{(r-3t)^{1/m}}\right]^{1/(m+3)}$

(d) $\left[\dfrac{(a+3)b^{n+k}}{d/(b-a)^{n+m}}\right]^{1/(k-2)}$

(e) $\dfrac{(x_1^2+x_2^3)^4(\sqrt{y_1}+\sqrt{y_2})^3}{(x_1/y_1)^{1/3}(x_2/y_2)^{1/4}}$

(f) $\dfrac{(a+3)b^n}{2.7(c-\dfrac{d}{b})+1}$

(g) $U=\dfrac{1}{2}e^{x(\sin x)^2}+y\ln(x^2+y^2)^{1/m}$

(h) $L=1+\dfrac{4x^2}{(2k^2+1)z^2}$

(i) $\left[\dfrac{(a/b)^n(c-d)^m}{d/(b-a)^{m+n}}\right]^{1/(m+n)}$

(j) $\left(\dfrac{\left(x-\dfrac{y}{z}\right)^{5/4}\left(1-\dfrac{x^2}{y^2+z^2}\right)}{\sqrt{\left(\dfrac{x+y}{x-y}\right)^3}}\right)$

(k) $\dfrac{\tan^{-1}x\sin^{-1}x}{\cosh^{-1}x-\sinh^{-1}x}$

2.8 Aşağıda verilen aritmetik ifadelerin formül hallerini çıkarınız.

```
(a) X=Y*(Z-D/E/2.)
(b) Y=X**3+5.*X**2+6.0
(c) X=Y+W/Z*5.
(d) P=P+R*P*T-1.25/P**3
(e) D=Dl+A*Z**3/3.
(f) K=3/(I/J-M)*L-NM-19
```

```
(g) I=J**(K-2)**L/7
(h) X=(1.-X*X)/(Y*Y-X*X)
(ı) Z=2.*A+3.*B/C**2-(1.+R)**N/M
(i) H=(1.+P)**3*(1.-Q)**.5/S*T/R
(j) H=2.*V**2*SIN(T)*COS(T-A)/(G*COS(T)**2)
(k) Z=1./(S*SQRT(2.*P))*EXP(-0.5*((X-A)**2/S**2))
(l) Z=(6.8*(A+B)**2/C-7.123*A/(B+C)**.5)/(A+C)**(1./N)
(m) H=(2.*A*B/(C+1.0)-T/(3.*(P+Q)))**(l./3.)
(n) H=(A+B*COS(X))/SIN(X*(A+B*SIN(X)))
(o) K=SQRT(SQRT(X**2+Y**2)+B*COS(A*X))/SQRT(X**2+Y**2)
(ö) Z=((2.*A*B/(C+1.))-(R/(7.*(P+Q))))**(1./N)
(p) H=R*A*(l.+R)**N/((l.+R)**N-1.0)
```

2.9 Aşağıdaki ifadelerde varsa hatalı olanlarını, hata nedenleri ile beraber tespit ediniz.

```
(a) X=4,562.23 (b) U=1.7687E-2.3       (c) T=53
(d) B=2.e3     (e) 5AB=7.94            (f) KL=2./5.
(h) Z=1511.    (i) ML=17               (j) T=1.+R**N
(k) X=X**N     (1) C=X**-2.            (m) E=(-4.)**.5
(n) C3=NM      (o) F=125./5./5./5.     (p) R=A*B/C*D/E
(q) Z=Z+1./Z   (r) W=(-2.4)**(-1.1)    (s) U=(1/(X-1/X))
```

2.10 Aşağıdaki programlarda derleme hatalarını bulunuz.

```
(a)  CHARACTER :: IKI*2,  BIR*1, SIFIR*0
     IKI='2'
     BIR='l'
     SIFIR=' '

(b)  N=5
     L=4
     CHARACTER :: A*N,  B*L

(c)  INTEGER :: X
     X=75
     REAL    :: Y
     Y=44.

(d)  INTEGER:: A; REAL:: Y,U,K; CHARACTER(LEN=3):: D,C

(e)  INTEGER :: Y
     REAL    :: X
     INT EGER:: A
     Y=3.25
     X=15
     A=7
```

2.11 Kütlesi 900 kg olan bir otomobil, 50, 80 ve 120 km/h sabit hızlarında hareket ettiğinde, otomobilin kinetik enerjisini kilo Joule birimi cinsinden hesaplayan bir program yazınız.

2.12 Bir dikdörtgensel kutunun ebadı 5 cm × l0.5 cm × 6.65 cm olarak veriliyor. Bu kutunun (a) hacmini (b) toplam yüzey alanını hesaplayan bir program yazınız.

2.13 Kartezyen koordinat sisteminde seçilen $P_1(x_1, y_1, z_1)$ ve $P_2(x_2, y_2, z_2)$ gibi iki noktanın bir birine olan uzaklığı, `d`, hesaplayan bir program yazınız. İki nokta arasındaki uzaklık aşağıdaki bağıntı ile verilmektedir:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

2.14 Bir cismin doğrusal genleşme katsayısı, birim sıcaklık değişimiyle cismin uzunluğunda meydana gelen değişim olarak tanımlanıyor. Bir çelik çubuk için yapılan ölçümler şöyle verilmektedir: 40°C'de uzunluk 120 cm iken sıcaklık 854°C'a çıkarıldığında 121.38 cm olarak saptanıyor. Çeliğin doğrusal genleşme katsayısını hesaplayan bir program yazınız.

2.15 Bir market sipariş verdiği 418 tür ürünü l ile 418 arasında kodlar (`Urun_Kodu`) vererek tanımlıyor. Yazacağınız bilgisayar programı sırasıyla ürünün kodunu, kaç adet satın alınacağı (`Miktar`) ve birim fiyatını (`Birim_Fiyat`) okuyan ve sipariş verilen ürünlerin bedelini (`Bedel`) hesaplayan bir program yazın. Programda kullanacağınız değişken isimleri parantez içinde belirtilmiştir.

2.16 Bir sarkacın periyodu

$$T = 2\pi\sqrt{\frac{L}{g}}\left(1 + \frac{1}{4}\sin^2\frac{\alpha}{2}\right)$$

formülü ile veriliyor. Burada g=980 cm/s^2, L ise sarkacın uzunluğu (cm) ve α yer değiştirme açısıdır. Girilen herhangi bir yer değiştirme açısı ve sarkaç uzunluğu için periyodu hesaplayan bir program yazınız.

2.17 Bir dikdörtgenin köşegenine göre atalet momenti $6(a^2 + b^2)$ olarak veriliyor. Burada a ve b dikdörtgenin kenarları olduğuna göre, ebadı 4 cm × 7 cm olan bir dikdörtgenin köşegenine göre atalet momentini hesaplayan bir FORTRAN programı yazınız.

2.18 Fizik derslerinin başında bahsedilen standard problemlerden biri de yayın ucuna yerleştirilen bir cismin basit harmonik hareketinin enerji hesabını içerir. Enerji terimleri kinetik, potansiyel ve toplam enerjidir. Kullanılan formüller, türetilmesini kapsayan detaylara girmeden, aşağıda verilmektedir. Bu hareket için bir FORTRAN programı yazınız.

$$K = \frac{4\pi^2 M}{T^2} \qquad TE = \frac{1}{2}K\,A^2 \qquad PE = \frac{1}{2}K\,X^2 \qquad KE = TE - PE$$

Burada K yay sabitini (kg/s^2), TE toplam enerjiyi (J), PE potansiyel enerjiyi (J), KE kinetik enerjiyi (J), A hareketin genliğini (m), X göz önüne alınan yer değiştirme miktarını (m), M cismin kütlesini (kg) ve T hareketin periyodunu (s) temsil etmektedir. Hareketin periyodu 0.5s, genliği 0.4m, kütlesi 6 kg ve yer değiştirmesi 20 cm olan durum için yay sabitini, PE, KE ve TE'yi hesaplatınız.

2.19 Aşağıdaki programlar çalıştırıldığında, sabit ve değişkenlerin bellek değerleri ne olur?

(a)
```
PROGRAM a
IMPLICIT NONE
REAL :: A, B, C, D, E
A=2.34
B=4.32
C=B; D=A; E=C
C=5.25
END PROGRAM a
```

(b)
```
PROGRAM b
IMPLICIT NONE
INTEGER :: J, M2, I, KJ, N2, M=9
J=M-1
M2=2
I=J
KJ=M2+I
M=N2
END PROGRAM b
```

(c)
```
PROGRAM c
IMPLICIT NONE
REAL    :: P=1.0, X, Y
INTEGER :: K, M, J
K=K+1
M=K
X=1.0+3.0*P
J=M+3
Y=P*X
END PROGRAM c
```

2.20 Aşağıdaki programların çıktılarını bulunuz.

(a)
```
PROGRAM Soru_a
IMPLICIT NONE
INTEGER :: I=2, J=5, K, L
K=I+J
L=K+I
I=L+1
L=K/I
PRINT*, I, J, K, L
END PROGRAM Soru_a
```

(b)
```
PROGRAM Soru_b
IMPLICIT NONE
REAL :: R=0.5, P=1500., T, X
INTEGER :: M=5
T=1.+R/10.
T=T**M
X=P*R/T
PRINT*, P, R, M, X
END PROGRAM Soru_b
```

(c)
```
PROGRAM Soru_c
IMPLICIT NONE
INTEGER :: I=2, J=3, K=4, M, N, NN, MM
L=I**J
M=I**(-J)
N=(J**I)**K
NN=J**(I*K)
MM=J**(I**K)
PRINT*, L, M, N, NN, MM
END PROGRAM Soru_c
```

(d)
```
PROGRAM Soru4
IMPLICIT NONE
REAL :: U=1.5, X, Y
INTEGER :: M=5
X=U+(-U)**(1./M)
Y=M*X
END PROGRAM Soru4
```

BÖLÜM 3
TEMEL PROGRAMLAMA TEKNİKLERİ

Programlama ve programlama tekniklerinden bahsetmeden önce "*Neden program hazırlama gereği duymaktayız?*" sorusunu soralım ve bu sorunun cevabını vermeye çalışalım. Öncelikle bilgisayarlar aritmetiksel işlemleri yaparken çok hızlı çalışırlar. Bir çarpma işlemi yaklaşık olarak l, üst alma işlemi yaklaşık 20 mikro saniyelik mertebelerde yapılır. Her gün yeni gelişen bilgisayarlarla bu süre daha da kısalmaktadır. İnsan beyni aritmetik işlemleri bu kadar hızlı yapamaz. Ayrıca milyonlarca kişinin ismini içeren bir dosyadan belirli bir ismin bulunması gibi bir işlemi de insandan daha hızlı yapabilmektedir. İnsan gücü ve kapasitesi bu problemleri çözmesi için yeterli olmasına rağmen çözüm süresiyle sınırlıdır. İnsanların haftalar süren çalışması sonucu çözdüğü problemi bir bilgisayar bir kaç dakika ile bir kaç saat arasında bir sürede çözebilmektedir.

Bilimsel ve mühendislik problemlerinin çözümünde binlerce, hatta milyonlarca, aritmetik işlemin yapılması olağandır. Bir optimum çözüm bulmak için problemi defalarca elle çözmenin çok vakit alacağı, ve hatta işlem hatası olasılığının mevcudiyeti, çok aşikar bir gerçektir. Bazı problemleri elle çözmek mümkün bile değildir. Bu nedenle program yazmak ihtiyacı duyulur ve işlemler bilgisayarla yaptırılır.

Bir bilimsel ve/veya mühendislik probleminin bilgisayar kullanarak çözümünde izlenmesi gereken belirli aşamalar vardır. Bunlar sırasıyla:

1. **Modelleme:** Fiziksel sistemi, bir ideal matematiksel modele çevirmek ve bu modele göre gerekli matematiksel denklemleri formüle etmek,

2. **Sayısal Yöntem Seçme:** Bilgisayarla programlamaya uygun bir nümerik yöntem seçmek (türev, interpolasyon, integral, denklem çözümünü bilgisayar programlama ile gerçekleştirmek için kullanılan yöntemler analitik yöntemlerden farklılık gösterir),

3. **Algoritma Oluşturma:** Yeni bir çözüm tekniği ortaya konuyorsa, bu tekniğin adımlarını veya aşamalarını belirtmek; (her aşamada yapılacak işlerin sıralı olarak belirtildiği düzene *algoritma* veya *program dizaynı* denir),

4. **Akış Şeması Oluşturma:** Detaylı bir akış şeması oluşturmak (yani *Oku*, *Yaz*, *Kıyasla*, *Hesapla* gibi işlemlerin sırasını belirleyen bir grafiksel gösterim şeklidir); akış şeması genellikle algoritma esas alınarak hazırlanır,

5. **Programlama Dilinde Kodlama:** Akış semasındaki işlemlere göre makinenin anlayacağı ve kabul edeceği bir dilde bu işlemleri yazmak veya diğer bir deyişle kodlamak (BASIC, PASCAL, FORTRAN, C gibi dillerle),

6. **Programı Test Etme:** Programın doğruluğunu test etmek amacıyla çalıştırmak; arzulanan sonucu vermiyorsa, programı kontrol etmek, hata(ları) bulup düzeltmek.

Bu aşamalar takip edildikten sonra, istenilen işlemleri yerine getirecek program hazırdır.

3.1 ALGORİTMA VE AKIŞ ŞEMASI

Akış şeması, verilen bir problemin çözümü için izlenmesi gereken yolun şekillerle gösterildiği bir şemadır. Çözülecek problemin matematiksel ve mantıksal işlemlerinin genel bir resmini sunar; çok uzun ve karmaşık problemlerin çözümünde algoritma hazırlamak kaçınılmazdır. Algoritma hazırlamak, hatası az olan program yazımına son derece önemli katkıda bulunur; iyi hazırlanmış bir algoritma, bazen akış şeması gerektirmeyebilir.

Bir problem için nasıl algoritma oluşturulur? Örneğin, bir arabanın çalıştırılması için bir algoritmanın nasıl hazırlanacağını inceleyelim.

1. Arabaya girip koltuğa otur.
2. Vitesi boşa al.
3. Viteste iken ayağınız debriyajda mı? Hayır ise (2.)'ye git.
4. Kontak anahtarını yuvasına koyduktan sonra çevirerek arabayı çalıştır
5. Gaz pedalına bas (gaz ver).
6. Debriyaja basarak vitesi 1. konumuna al.
7. Hafif hafif gaz verirken ayağı debriyajdan çek.
8. Bir kaç araba boyu ilerledikten sonra debriyaja basarak vites yükselt.
9. Gaz vererek hızını artır.
10. Motorun devri artmış ise, debriyaja bas vites yükselt. (9.)'ye git.
11. Motorun devri düşmüş ise, debriyaja bas vites küçült.
12. Hızlanmak için gaz ver, yavaşlamak için hafif fren yap.
13. Durmak için debriyaj ve fren pedallarına beraber basarak durdur.
14. El frenini çek ve kontağı kapat.

Buradaki işlem sırasını takip etmek zorundayız; bunu yapmadığımız takdirde arabamız çalışmayacak ya da tuhaf bir şekilde davranacak ve arabamıza hasar verebileceğiz. Şimdi de, bir bilgisayarın arabayı çalıştırmasını ve bunun için bilgisayarı programlamak istersek, algoritmamız bu kadar basit olmayacaktır; çünkü bir insanın araba kullanırken karşısına çıkacak her türlü olasılığı, bilgisayara program ile bildirmek gerekecektir. Örneğin, hava yağışlı ise sileceklerin, sıcak ise klimanın, soğuk ise ısıtıcının çalıştırılması; yol buzlu ise fazla hız yapılmaması, sağa ve sola dönüşlerde sinyal verilmesi, gece kullanımında farların yakılması gibi daha binlerce olasılığı düşünmek ve programlamak gerekir. Bu programın girdi verilerini görüş açısı, yakıt ve hız göstergeleri, sinyal göstergesi v.s oluşturmakta iken, program çıktısı arabanın hareketi, davranışı ve göstergelerdeki değişimlerdir.

Araba kullanma olayını bir robota yaptırmaya kalkacak olursak, robota hükmedecek program çok daha karmaşık olacaktır. Robotun dikiz aynalarına bakması, görüntüyü algılaması ve yorumlaması, direksiyonu kavraması, ayağının pedallara basma hızı v.b daha nice olasılıklarını programcı göz önüne almak zorundadır.

Kısacası bir program yazmadan önce amacımıza ulaşmak, karşımıza çıkacak her türlü soruyu çözmek için bilgisayara önceden direktif vermeliyiz; aksi takdirde program bu durumlarda ne yapacağını bilemez. Unutmayınız, bilgisayarlar sizin ne düşündüğünüzü, ne yapmak

istediğinizi bilmezler; onlar sadece programlanan koşullara ve olasılıklara tepki verirler; dolayısıyla, bilgisayar ancak programı hazırlayanlar kadar zekidir.

3.2 AKIŞ ŞEMASI

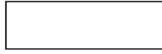
Akış şeması, bir program hazırlanırken programcıya ya da programı inceleyene kolaylık sağlayan bir grafiksel gösterim yoludur. Programcılar akış şemasına bakarak istenilen herhangi bir programlama dilini kullanarak programlama yapabilirler. Diğer bir deyişle, akış şeması programlama dilleri söz konusu olduğunda geneldir.

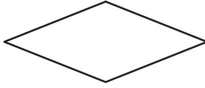
Akış şemasında kullanılan geometrik şekiller aşağıda verilmiştir.

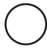

Başlama ve bitiş durumlarını gösterir, içine BAŞLA veya DUR yazılarak kullanılır.


Giriş-Çıkış işlemlerinde kullanılır; OKU < değişken listesi>, ve YAZ <değişken listesi> şeklinde kullanılır.

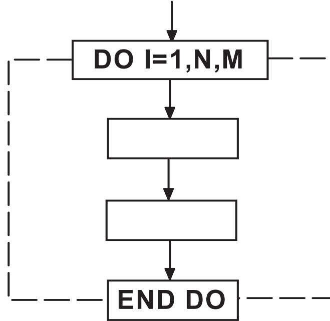

Akış yönünü gösterir.


İşlem ünitesidir. Aritmetik işlemler için kullanılır


Karar aşamasını belirtir

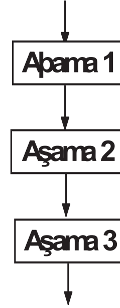

Akış şeması bir sayfaya sığmadığında veya bir altprograma atıfta bulunulduğunda içine bir numara veya harf konularak kullanılır.

Kapalı döngü (`DO-END DO`) belirtir.

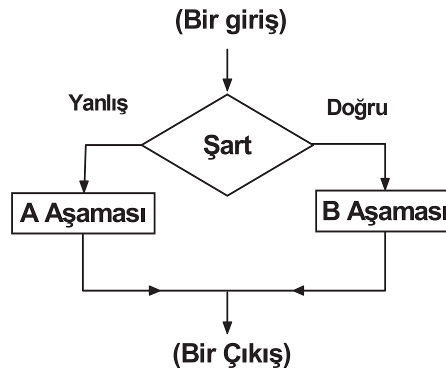


Akış şemalarında karşılaşılan durumlara şu örnekler verilebilir.

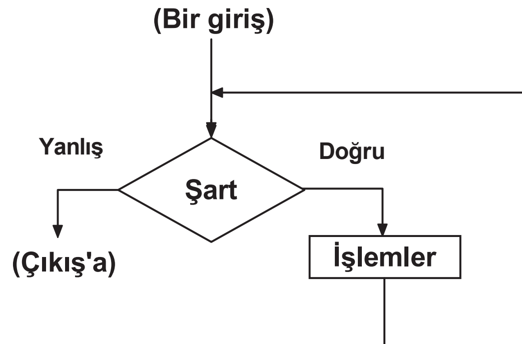
(i) Sıralı Akış: Bir birini takip eden birden fazla aritmetik işlem aşamalarının olması durumudur.



(ii) Şartlı Akış: Bir şartlı bileşeni içermesi durumudur



(iii) Tekrarlı Akış: Bir işlemin veya işlemler dizisinin belirli sayıda işlem görmesi



ÖRNEK 1: Bir ürünün toptancı fiyatı (`Fiyat`) ise, aracı kişi bunu satın alınca % 15 KDV ödemekte, kendisi de % 25 kar payını da koyduktan sonra KDV'siyle beraber tüketiciye satmaktadır. Bu durum için bir algoritma aşağıdaki şekilde hazırlanabilir.

1. Malın toptancı fiyatını (`Fiyat`) gir.

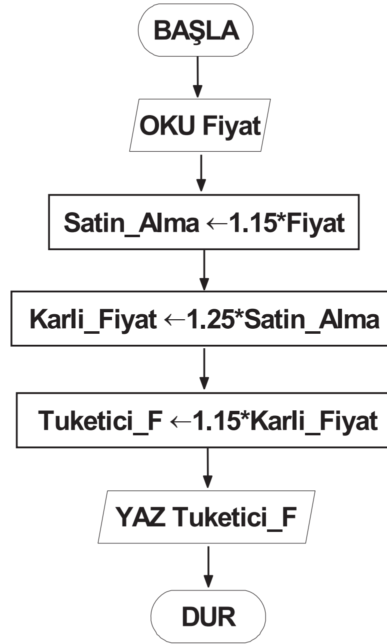
2. Aracının satın alma fiyatını (`Satin_Alma`) hesapla.

3. Aracı kârıyla beraber fiyatı (`Karli_Fiyat`) hesapla.

4. Tüketiciye ulaşan fiyatı (`Tuketici_F`) hesapla.

5. Çıktıyı, `Tuketici_F`, ekrana yaz.

6. Dur.



ÖRNEK 2: Şartlı akış durumuna bir örnek olarak pozitif ve negatif sayıların tespitini ele alalım.

1. Ekrandan bir sayı (**P**) oku.

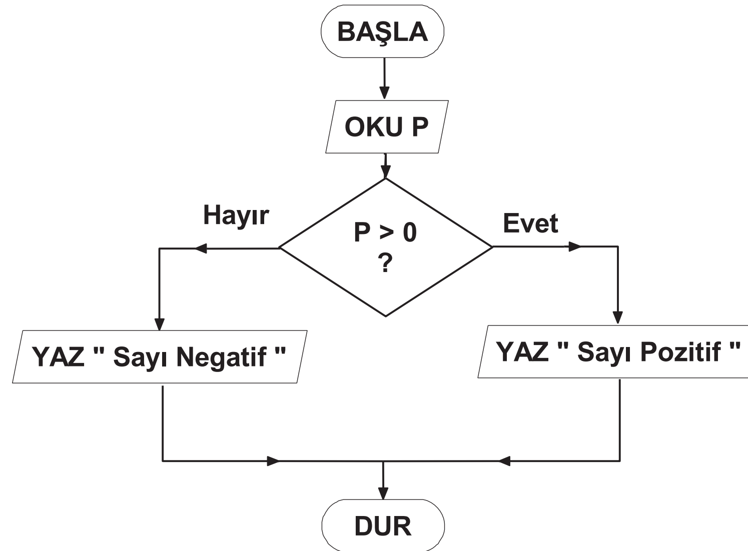
2. Eğer sayı pozitif ise (5.) aşamaya (**P>0**) git (negatif ise devam)

3. Ekrana "`Sayı negatiftir`" yaz.

4. (6.cı) aşamaya git

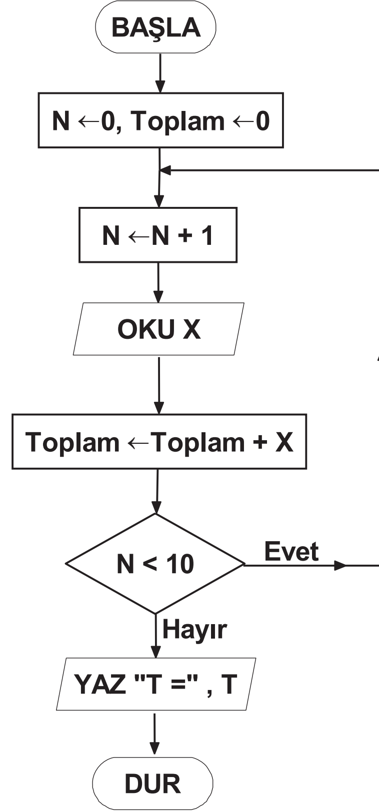
5. Ekrana "`Sayı pozitiftir`" yaz.

6. Dur.



ÖRNEK 3: Tekrarlı akış durumuna bir örnek olarak, programa dışarıdan girilecek 10 sayının toplamım ele alalım:

1. Sayaç (`N`) ve toplamı (`Toplam`) sıfırla (`N`←0 ve `Toplam`←0).
2. Sayacın değerini bir artır.
3. `X` değerini oku.
4. X değerini toplama kat. `Toplam←Toplam+X`
5. Eğer `N`, l0'dan küçük ise (2.) aşamaya git.
6. `T` değerini ekrana yaz.
7. Dur.



3.3 KARAR VERME YAPISI

Aritmetik işlemlerin yanısıra ALU (Arithmetic-Logic Unit-Aritmetik Mantık Birimi) da iki unsuru kıyaslamak ve kıyasa dayanan komutları yürütmek için tasarlanmıştır. Bu kıyaslamalardan en basiti, iki unsurun aynı olup olmamasını içerir. Bu kıyaslamanın FORTRAN'da kolaylıkla yapılabilmesi için mantıksal ifadelere yer verilmektedir. Mantıksal ifadeler, mantıksal IF testlerinde ve mantıksal blok IF yapılarında kullanılır. Mantıksal IF testi, mantıksal blok IF testinin kısa bir şeklidir.

3.3.1 MANTIKSAL İFADELER

`IF` deyiminin kullanımı tabi ki mantıksal ifadenin tanımına bağlıdır. Bir mantıksal ifade bir veya daha fazla ilişkisel ifadenin kullanımını içerebilir. Bu ifadeler birbirlerine bir "*ilişkisel mantık operatörler*" ile bağlanırlar. Bunlar:

Operasyon		*Anlamı*
Yeni	*Eski*	
`==`	`.EQ.`	Eşit
`/=`	`.NE.`	Eşit değil
`>`	`.GT.`	Büyük
`>=`	`.GE.`	Büyük eşit
`<`	`.LT.`	Küçük
`<=`	`.LE.`	Küçük eşit

Örneğin, `25.LT.98` (`25<98`) ifadesinin değeri `TRUE` dur. Diğer taraftan, program içinde kullanıma örnek olarak

Bazı ilişkisel işlemler ve sonuçları aşağıda verilmiştir:

İşlem	*Sonuç*
`3 < 4`	`.TRUE.`
`3 <= 4`	`.TRUE.`
`3==4`	`.FALSE.`
`3 > 4`	`.FALSE.`
`4 <= 4`	`.TRUE.`
`'A'<'B'`	`.TRUE.`

Örneğin `X` değişkeninin değeri `2.` ise `X<5.0` mantıksal ifadesinin (önermesinin) sonucu TRUE'dur. Bu tür mantıksal önermelerde aritmetiksel işlemler mevcut ise mantıksal işlemlerden aritmetik işlemler yapılır, daha sonra önermenin doğru veya yanlış olduğuna bakılır. Örneğin, `(X<2.5*SQRT(10.))` ifadesinde önce $2.5\sqrt{10}$ hesaplanır ve `X` ile kıyaslanır; bu önermenin değeri `TRUE` bulunur.

Önermelerde fazla sayıda parantezlerin kullanımı, `(ABS(X-3.)<(10.-SQRT(5.))` gibi, ifadeye açıklık kazandırmak bakımından ilave edilebilir.



Mantıksal önermelerde önce "aritmetik işlemler" yapılır, sonra mantıksal aritmetiğe geçilir.

Birleştirici mantıksal operatörlerin kullanımıyla daha karmaşık mantıksal ifadeler oluşturulabilir.

Tablo 3.1 Birleştirici mantıksal bağlaçların işlevleri ve tanımları.

OPERATÖR	FONKSİYONU	TANIMI
ℓ_1 **.AND.** ℓ_2	Mantıksal **AND** (VE bağlacı)	ℓ_1 ve ℓ_2'nin her ikisi de .TRUE. ise önerme sonucu .TRUE.'dur.
ℓ_1 **.OR.** ℓ_2	Mantıksal **OR** (VEYA bağlacı)	ℓ_1 veya ℓ_2'den en az biri .TRUE. ise önerme sonucu .TRUE.'dur.
ℓ_1 **.EQV.** ℓ_2	Mantıksal **EQV**—denklik (DENK bağlacı)	ℓ_1 ve ℓ_2'nin her ikisi de aynı değeri alıyorsa (örneğin, ℓ_1 = .FALSE. ve ℓ_2 = .FALSE.) önerme .TRUE.'dur.
ℓ_1 **.NEQV.** ℓ_2	Mantıksal **NEQV** (DENK DEĞİL bağlacı)	ℓ_1 ve ℓ_2'den birisi .TRUE. ve diğerisi de .FALSE. ise önerme .TRUE.'dur.
.NOT. ℓ_1	Mantıksal **NOT** (DEĞİL bağlacı)	ℓ_1 .TRUE. ise değerini .FALSE. ya da .FALSE. ise .TRUE. yapar.

Tablo 3.2 Birleştirici mantıksal bağlaçların kullanımını veren tablo.

ℓ_1	ℓ_2	ℓ_1 **.AND.** ℓ_2	ℓ_1 **.OR.** ℓ_2	ℓ_1 **.EQV.** ℓ_2	ℓ_1 **.NEQV.** ℓ_2
.FALSE.	.FALSE.	.FALSE.	.FALSE.	.TRUE.	.FALSE.
.FALSE.	.TRUE.	.FALSE.	.TRUE.	.FALSE.	.TRUE.
.TRUE.	.FALSE.	.FALSE.	.TRUE.	.FALSE.	.TRUE.
.TRUE.	.TRUE.	.TRUE.	.TRUE.	.TRUE.	.FALSE.

Mantıksal ifadeler **.AND.**, **.OR.**, **.NOT.**, **.EQV.** ve **.NEQV.** operatörlerinin kullanımıyla birleştirilerek yeni önermeler oluşturulabilir. Bu operatörlerin uyduğu kurallar ve fonksiyonu Tablo 3.2 de verilmektedir. Örneğin, X ve Y sırasıyla 1. ve 5. değerlerine sahip olsunlar, bu durumda (X>3.0.**AND**.3.*Y<30.) ifadesi (*false*.AND.*true*) olacak; ve sonuçta önerme false olarak değerlendirilecektir. Ayrıca, (X*Y==0.0.**OR**.Y<13.) ifadesi (false.**OR**.true) olacak, sonuçta *true* değerini alacaktır.



Mantıksal önermelerle işlemlerde aritmetik işlemler işlem hiyerarşisine uygun olarak hesaplanır. Mantıksal operatörler soldan sağa doğru taranarak değerlendirilir.

Örneğin, (I==10.**OR**.X<1.0.**AND**.Z>=0.0) olarak verilen bir birleşik önerme esasında derleyici tarafından ((I==10).**OR**.(X<1.).**AND**.(Z>=0.0))şeklinde yorumlanmaktadır ve ((I==10).OR.((X<1.).AND.(Z>=0.0))) şeklindeki yazılım ile eş değerdir. Eğer I=10, X=0.0 ve Z=-1.0 ise, bu durumda yukarıdaki önerme

(*true*.**OR**.*true*.**AND**.*false*) olarak okunacak, *true*.**AND**.*false*=false ve bunun sonucunda da (*true*.**OR**.*false*) önermesine eşdeğer olacak; sonuç olarak bu önerme *true* değerini alacaktır.

3.3.2 BLOK IF YAPISI

Sıklıkla kullanılan **IF** koşul yapılarından biridir. **IF** ile belirlenen mantıksal ifadenin "doğru" olduğu durumda gerçekleştirilecek blok deyimlerin yer aldığı bir yapıdır. Mantıksal ifade "yanlış" ise derleyici bu satırı atlayarak **END IF**'den sonraki satıra geçer. Genel kullanım şekli aşağıda verilmiştir.

```
IF (mantıksal_önerme) THEN
            Deyim_1  }
            Deyim_2  }  Blok kısmı
            Deyim_3  }
END IF
```

Aşağıdaki örneklerden görüleceği üzere IF koşulu gerçekleştiğinde **IF-END IF** bloğu içinde belirlenmiş deyimler göz önüne alınmaktadır.

```
(a)  IF(A>B) THEN
          C=A*A+B*B
          D=TAN(C/(A+B))
          E=D**C
     END IF
```

```
(b)  IF(T<=3) THEN
          Z=SIN(3.14*T)
          X=COS(3.14*T)
     END IF
```

```
(c)  IF(b*b-4.*a*c<0.) THEN
        PRINT*, 'İkinci dereceden denklemin kökleri sanaldır'
     END IF
```

3.3.3 IF-ELSE-END IF YAPISI

Bir algoritmada mantıksal IF testinin iki sonucu vardır; şartın TRUE ve FALSE olduğu durumlar için birer Fortran deyimleri blokları kullanılır. Her iki şarta cevap verme ihtiyacının olduğu böylesi durumlarda IF-THEN-ELSE-ENDIF yapısı kullanılır. Bütün karar verme yapılarının temel taşı olan mantıksal IF yapısının genel kullanım şekli aşağıda verilmektedir.

```
IF ( <mantıksal önerme> ) THEN
     BLOK 1
     Sadece mantıksal ifadenin doğru (.TRUE.) olduğu durumda işlenecek
     Fortran deyimleri gurubu (blok deyimler) yer alır.
ELSE
     BLOK 2
     Sadece mantıksal ifadenin doğru (.TRUE.) olduğu durumda işlenecek
END IF
```

Mantıksal önerme "*doğru*" ise birinci, "*yanlış*" ise ikinci blokta yer alan işlemler dizisi gerçekleştirilir.
Örneğin,

```
Delta=b*b-4.0*a*c
IF(Delta>=0.0) THEN
     d=SQRT(Delta)
     PRINT*, "İkinci dereceden denklemin kökleri"
     X1=(-b-d)/(2.*a)
     X2=(-b+d)/(2.*a)
     PRINT*, "X1=",X1, " X2=",X2
ELSE
     PRINT*, "Denklemin kökleri sanaldır"
ENDIF
```

Bu örnekte kullanılan `IF-THEN-ELSE-ENDIF` bloğunun içinde yer alan işlemlerin, bloğu belirginleştirmek için, beş karakter içeriden yazıldığına dikkat ediniz.



Birden fazla iç içe `IF-THEN-ELSE-ENDIF` bloğu kullanıldığında, programda okunabilirlik ve takip edebilirlik sağlamak için blok deyimlerini, örneklerdeki gibi, biraz içinden yazınız.

3.3.4 ELSE VE ELSE IF KULLANIMI

Bir dizi koşulun bulunduğu durumda farklı koşullarda yerine getirilecek işlemlerin bir arada verilmesi bakımından yararlıdır.

```
IF (<mantıksal önerme_1>) THEN
    Mantıksal önerme_1'in doğru (.TRUE.) olduğu durumda işlenecek blok
     Deyim_11
     Deyim_21
       . . .
       . . .
ELSE IF (<mantıksal önerme_2>) THEN
    Mantıksal önerme_2'nin doğru (.TRUE.) ve mantıksal önerme_1'in yanlış
     (.FALSE.) olduğu durumda işlenecek blok
     Deyim_21
     Deyim_22
     . . .
     . . .
ELSE
    Mantıksal önerme_2'nin yalnış (.FALSE.) ve mantıksal önerme_1'in yanlış
     (.FALSE.) olduğu durumda işlenecek blok
     Deyim_31
     Deyim_32
     . . .
     . . .
END IF
```

Örneğin,

```
PROGRAM ikinci_derece_denklem
IMPLICIT NONE
REAL :: a, b, c, d, Delta, x1, x2
READ*, a, b, c
Delta=b*b-4.0*a*c
IF(Delta>0.0) THEN              ! Delta>0 Bloğu
     d=SQRT(Delta)
     PRINT*, "İkinci dereceden denklemin kökleri"
     x1=(-b-d)/(2.*a)
     x2=(-b+d)/(2.*a)
     PRINT*, "x1=",x1," x2=",x2
ELSE IF(ABS(Delta)<1.e-5) THEN ! Delta = 0 Bloğu
     PRINT*, "Kökler eşittir"
     x1=-b/(2.*a)
     PRINT*, "x1=x2=",x1
ELSE                           ! Delta < 0 Bloğu
     PRINT*, "Denklemin kökleri sanaldır"
ENDIF
END PROGRAM ikinci_derece_denklem
```

ÖRNEK 4: Aşağıdaki şekilde verilen bir fonksiyonu, dışarıdan girilen herhangi bir *x* değeri için hesaplayan bir FORTRAN programı yazınız.

$$f(x) = \begin{cases} 2.5x + \pi, & x < 0; \\ x^2 - 4, & 0 \le x \le 5; \\ \sin x - \cos x, & 5 < x \le 7; \\ \pi & x > 7. \end{cases}$$

`IF-THEN-ELSE IF-ENDIF` yapısını kullanarak aşağıdaki şekilde programlayabiliriz. Her bir koşulun gerçekleştiği durumda fonksiyonun hesaplandığı işlemler atama deyimleri ile gerçekleştirilir.

```
PROGRAM fonksiyon
IMPLICIT NONE
REAL :: x, F, pi=3.14159
PRINT*, 'x Değerini Giriniz'; READ*,  x
IF(x<0.0)THEN
     F=2.5*x+pi
ELSE IF(x>=0.0.AND.x<=5.0) THEN
     F=x*x-4.0
ELSE IF(x>5.0.AND.x<=7.0) THEN
     F=SIN(x)-COS(x)
ELSE
      F=pi
ENDIF
PRINT*, 'F(',x,')=',F
END PROGRAM fonksiyon
```

3.3.5 İSİMLİ BLOK IF KULLANIMI

IF bloklarına isim vermek mümkündür. İsim bir alfabe karakteri ile başlayıp, en fazla 31 karakter uzunluğunda olabilir. Kullanılan isim END IF deyiminden sonra da yazılmalıdır. IF bloğu için kullanılan isim başka amaçlar için kullanılmamalıdır.

```
Dıstaki : IF (<mantıksal önerme_1>) THEN
              Deyim_11
              Deyim_21
              . . .
       icteki: IF (<mantıksal önerme_2>) THEN
                    Deyim_21
                     Deyim_22
                     . . .
               END IF icteki
               . . .
          END IF Distaki
```

Bir programda kullanılan IF-THEN (ELSE) yapıları iç-içe yuvalanmalıdır. Örneğin,

```
IF(A>=0.005)THEN
   ...
   ...
   IF(B<3.14) THEN
      ...
   ENDIF
   ...
   IF(n>=20) THEN
      ...
   ELSE
      ...
   ENDIF
   ...
ENDIF
```

```
IF(A>=0.005)THEN
   ...
      IF(B<3.14) THEN
         ...
      ENDIF
   ...
ELSE
   IF(n>=20) THEN
      ...
   ELSE
      ...
   ENDIF
   ...
ENDIF
```

şeklinde verilen yapılar "*doğru*" olarak yuvalanmış IF yapılarını göstermektedir. Çok sayıdaki IF deyimlerinin kullanılması halinde hangi deyim END IF ile kapatıldı takibi güçleşmektedir. Bu nedenle, bloklara isim vermek makul bir strateji olacaktır.

"*Yanlış*" yuvalanan IF yapılarına bazı örnekler de aşağıda verilmektedir.

```
if1 : IF (x<1.45) THEN              if1 : IF (x<1.45) THEN
   ...                                 ...
  if2 : IF (B>=999) THEN ┐          if2 : IF (B>=999) THEN ┐
     ...                 │             ...                 │
  ENDIF if1              │          ELSE (if1'e ait)       │
     ...                 │                ...              │
     ENDIF if2           ┘                ENDIF if2        ┘
                                       ...
                                    ENDIF if1
```

İki `IF` ile belirli bloklar birbirlerini kesmemelidir!

ÖRNEK 5: Bir sınıftaki öğrencilerin ham notlarına aşağıdaki şekilde harf notları atayacak programın *koşullu* kısmını yazınız.

95 < NOT	AA
87 < NOT ≤ 95	BA
79 < NOT ≤ 87	BB
71 < NOT ≤ 79	CB
67 < NOT ≤ 71	CC
59 < NOT ≤ 67	DC
49 < NOT ≤ 59	DD
49 ≤ NOT	FF

Bir not verilen not aralıklarına karşılık geldiğinde harf notunu yazan bir program hazırlayabiliriz. Bu amaçla not aralıklarını `IF-THEN-ELSE IF` yapısal deyimi ile belirlememiz daha kolay olmaktadır. Bu durumda

```
IF(NOT>95.) THEN
     PRINT*, "Harf Notu AA dır"
ELSE IF(NOT>87.) THEN
     PRINT*, "Harf Notu BA dır"
ELSE IF(NOT>79.) THEN
     PRINT*, "Harf Notu BB dir"
ELSE IF(NOT>71.) THEN
     PRINT*, "Harf Notu CB dır"
ELSE IF(NOT>67.) THEN
     PRINT*, "Harf Notu CC dir"
ELSE IF(NOT>59.) THEN
     PRINT*, "Harf Notu DC dir"
ELSE IF(NOT>49.) THEN
     PRINT*, "Harf Notu DD dır"
ELSE
     PRINT*, "Harf Notu FF dir"
END IF
```

`IF-ELSE IF` deyimlerinden fazla sayıda kullandığımızdan dolayı, bazı yanlışlıklar yapmanın önüne geçmek için, hangi `if` deyiminin nerede açıldığı ve nerede kapandığını

isimli yapı kullanarak görebiliriz. Bu şekilde isimli IF yapısı kullanıldığında, program

```
if1 : IF(NOT>95.) THEN
   PRINT*, "Harf Notu AA dır"
ELSE
   if2 : IF(NOT>87.) THEN
      PRINT*, "Harf Notu BA dır"
   ELSE
      if3 : IF(NOT>79.) THEN
         PRINT*, "Harf Notu BB dir"
      ELSE
         if4 : IF(NOT>71.) THEN
            PRINT*, "Harf Notu CB dır"
         ELSE
            if5 : IF(NOT>67.) THEN
               PRINT*, "Harf Notu CC dir"
            ELSE
               if6 : IF(NOT>59.) THEN
                   PRINT*, "Harf Notu DC dir"
               ELSE
                  if7 : IF(NOT>49.) THEN
                      PRINT*, "Harf Notu DD dır"
                  ELSE
                     PRINT*, "Harf Notu FF dir"
                  END IF if7
                END IF if6
              END IF if5
            END IF if4
          END IF if3
     END IF if2
END IF if1
```

şeklinde oluşturulabilir.

3.3.6 MANTIKSAL IF DEYİMİ

Bir koşula karşın sadece bir deyim icra edilecekse, bu işlemi `IF-END IF` bloğu kullanmak yerine sadece bir satırda mantıksal `IF` deyimi ile belirtmek mümkündür.

Eğer mantıksal önerme `FALSE` ise bir sonraki satıra geçilir.

```
IF (<mantıksal önerme>) Deyim
```

Örnekler,

```
IF(A<=B) T=B-A
IF(N>100) CYCLE
 . . .
```

```
IF(Delta<0) PRINT*, "Kökler sanaldır"
 . . .
IF(N==1) M=2*N-1
 . . .
IF(KOD>=90) EXIT
 . . .
IF( X<l.) Y=X*X*X-3.*X*X+5.
IF(X>=l.) Y=2.-X*X
```

3.4 KOŞULLU CASE YAPISI

Mantıksal `IF` yapısına alternatif bir diğer FORTRAN deyimi de **CASE** yapısıdır. Bir karakter, bir mantıksal ifade veya bir tam sayının değerine göre bir blok işlemleri yapar. Genel kullanım şekli

```
[isim  :] SELECT CASE (<ifade>)
            CASE (secim_1) [isim]
                 Deyim_11 )
                 Deyim_21 }   Blok 1
                   . . .  )
            CASE (secim_2) [isim]
                 Deyim_21 )
                 Deyim_22 }   Blok 2
                 . . .    )
            . . .
            CASE DEFAULT [isim]
                 Deyim_n1 )
                 Deyim_n2 }   Blok n
                 . . .    )
         END SELECT [isim]
```

olarak verilir. `IF` bloklarında olduğu gibi, `CASE` yapısında da isim kullanılabilir. Bazı örnekler,

```
(a)  PROGRAM Ornek_a
     IMPLICIT NONE
     INTEGER :: deger
     DO
     PRINT*, "Bir Sayı Giriniz"; READ*, deger
     SELECT CASE (deger)
          CASE (1,3,5,7,9)
               PRINT*, "Değer TEK SAYIDIR"
          CASE (2,4,6,8,10)
```

```
          PRINT*, "Değer ÇİFT SAYIDIR"
     CASE (11:)
          PRINT*, "Değer çok büyüktür"
     CASE DEFAULT
          PRINT*, "Değer sıfır yada negatiftir"
          EXIT
END SELECT
END DO
END PROGRAM Ornek_a
```

(b)
```
PROGRAM Ornek_b
IMPLICIT NONE
REAL, PARAMETER :: pi=3.14159
REAL :: a=10.
SELECT CASE (a*SQRT(pi))
     CASE (0:)
          PRINT*, "a>0"
     CASE (:0)
          PRINT*, "a<0"
     CASE DEFAULT
          PRINT*, "a=0"
END SELECT
END PROGRAM Ornek_b
```

(c)
```
PROGRAM Ornek_c
IMPLICIT NONE
CHARACTER(LEN=7) :: renk
READ*, renk
SELECT CASE (renk)
     CASE ("Kırmızı")
          PRINT*, "DUR"
     CASE ("Yeşil  ")
          PRINT*, "GEÇ"
     CASE ("Sarı    ")
          PRINT*, "HAZIRLAN"
     CASE DEFAULT
          PRINT*, "YANLIŞ GİRDİ İLE KARŞILAŞILDI! "
END SELECT
END PROGRAM Ornek_c
```

(d)
```
PROGRAM Ornek_a
IMPLICIT NONE
INTEGER :: I
READ*, I
SELECT CASE (I)
     CASE(1); PRINT*, "I=1"
     CASE(2:9); PRINT*, "I>=2 ve I<=9"
     CASE(10); PRINT*, "I>=10"
     CASE DEFAULT; PRINT*, "I<0"
END SELECT CASE
END PROGRAM Ornek_d
```

ÖRNEK 6: CASE yapısını kullanarak iki tamsayıya dört aritmetik işlemi uygulayan bir hesap makinesi programı yazınız.

Bu amaçla +, -, * ve / işlemlerini karakter olarak tanımlayıp, her işarete göre iki sayıya uygulanacak aritmetik işlemi belirleyebiliriz. İşlemi bir sorgulama ve hesap işlemini bir DO döngüsü içerisinde ve CASE yapısı kullanmak suretiyle belirtelim. Hesap makinesinden çıkış işlemini de operatörün dört işlem karakterlerinden farklı bir karaktere bağlayacağız.

```
PROGRAM Hesap_Makinesi
IMPLICIT NONE
!
! Basit bir hesap makinesi işlevi gören program
!
INTEGER :: I,J,K
CHARACTER :: Operator
DO
     PRINT *,' İki Tamsayı giriniz'
     READ *, I,J
     PRINT *,' Operatörü (+, -, *, /) Giriniz'
     READ '(A)', Operator
     HesapMak : &
     SELECT CASE (Operator)
          CASE ('+') HesapMak
               K=I+J
               PRINT *,' Toplam = ',K
          CASE ('-') HesapMak
               K=I-J
               PRINT *,' Fark = ',K
          CASE ('/') HesapMak
               K=I/J
               PRINT *,' Bölüm = ',K
          CASE ('*') HesapMak
               K=I*J
               PRINT *,' Çarpım = ',K
          CASE DEFAULT HesapMak ! +, -, *, / değilse ÇIK
               EXIT
     END SELECT HesapMak
END DO
END PROGRAM Hesap_Makinesi
```

ÖRNEK 7: 80 karakterden oluşan bir satırda sesli ve sessiz harfleri, boşlukları ve rakamların sayısını hesaplayan bir program yazınız.

Bu örnekte de sesli, sessiz harfler, boşluklar, rakamlar ile bu guruba girmeyen diğer karakterleri saymak için CASE yapısından yararlanacağız. Bu nedenle her gurupta yer alan karakter özellikleri CASE (..) içinde ayrıntılı olarak vermeliyiz.

```
PROGRAM Ornek_7
IMPLICIT NONE
INTEGER :: Sesli=0 , Sessiz=0, Basamak=0
INTEGER :: Bos=0, Diger=0, I
CHARACTER :: Harf       ! Uzunluğu 1 karakter olarak algılanmaktadır.
CHARACTER (LEN=80) :: Satir
READ '(A)', Satir
     DO I=1,80           ! 1.ci sütundan 80.ci sütuna kadar sırayla karakterler
          Harf=Satir(I:I) ! I.ci sütünda yer alan karakteri harf'e atıyoruz.
          SELECT CASE (Harf)
               CASE('A','E','I','İ','O','Ö','U',&
                      'Ü','a','e','ı','i','o','ö', &
                      'u','ü')
                    Sesli=Sesli + 1
               CASE('B','C','Ç','D','F','G','H', &
                      'J','K','L','M','N','P','Q', &
                      'R','S','Ş','T','V','W','X', &
                      'Y','Z','b','c','ç','d','f', &
                      'g','h','j','k','l','m','n', &
                      'p','q','r','s','ş','t','v', &
                      'w','x','y','z')
                    Sessiz=Sessiz + 1
               CASE('1','2','3','4','5','6','7','8','9','0')
                    Basamak=Basamak + 1
               CASE(' ')
                    Bos  = Bos + 1
               CASE DEFAULT
                    Diger= Diger+1
          END SELECT
     END DO
     PRINT *, ' Sesli  =  ', Sesli
     PRINT *, ' Sessiz =  ', Sessiz
     PRINT *, ' Basamak=  ', Basamak
     PRINT *, ' Boşluk =  ', Bos
     PRINT *, ' Diğer karakterler = ', Diger
END PROGRAM Ornek_7
```

CASE içinde o blok içinde yer alan koşullar birden fazla ise virgül ile ayrılarak ('1','2','3' gibi) sıralanmaktadır. Burada kullanılan Satir(I:I) işleminin ayrıntısına ileride değinilecektir; ancak kısaca ne yaptığımızı belirtmek açısından açıklayacak olursak; satir alfa sayısal değişkeninin I.ci sütunundaki karakteri harf değişkenine atamaktadır. Örneğin, 'deneme' isimli bir alfa sayısal sabiti veya değişkeni için deneme(3:3)=n veya deneme(5:5)=m ve deneme(2:2)=deneme(4:4)=deneme(6:6)=e olmaktadır.

3.5 ŞARTLI DÖNGÜLERİN POTANSİYEL HATALI KULLANIM DURUMLARI

Bir mantıksal IF veya CASE yapısı kullanırken çok sık yapılan hatalara değinmekte yarar vardır. Mantıksal IF yapısını

1. Gerçek (REAL) sayılarla *eşitlik kıyaslaması* için kullanmayınız. Bunun nedeni gerçek sayıların yaklaşık değerlerden oluşmalarıdır. Örneğin,

```
REAL :: A, B
A=2.0; B=SQRT(A)**2
IF(A==B) THEN
   PRINT*, ' A, B"YE EŞİTTİR'
END IF
```

veya

```
SELECT CASE (a)
     CASE (a=3.15)
     . . .
     CASE (a==b)
END SELECT
```

ifadelerinde B tam olarak 2'ye eşit değildir ($B \neq 2$) çünkü $B = 2 \mp \varepsilon$ gibi bir değer almakta olup, bu kıyaslama *başarısızlık* ile sonuçlanacaktır.

2. Eğer bir gerçek değerin EPSILON gibi küçük bir sayıdan daha küçük olup olmadığı araştırılmak isteniyorsa, bu kıyaslamayı asla

```
REAL :: U, EPSILON=1.e-6
IF(U<EPSILON) THEN
     PRINT*, 'U, EPSILON DAN KÜÇÜKTÜR'
ENDIF
```

şeklinde *yapmayınız*! Bunu daha ziyade

```
IF(ABS(U)<EPSILON) THEN
     PRINT*, 'U, EPSILON DAN KÜÇÜKTÜR'
ENDIF
```

şeklinde kıyaslayınız. Çünkü U negatif değerlerde alabilir; bu durumda bir negatif değer, büyüklüğü ne olursa olsun, yukarıdaki koşulun ($U < \varepsilon$) sağlanmasına neden olur. Bu nedenle koşulu $|U| < \varepsilon$ şeklinde oluşturmak daha uygundur. Burada EPSILON gibi küçük bir sayının ne olması gerektiğini (10^{-3} yoksa 10^{-5} mi) algoritma gereği programcı belirler.

İki gerçek sayının (A ve B gibi) eşitliğinin kıyaslanması da, bu bilgilerin ışığı altında,

```
REAL :: A, B, EPSILON=1.e-6
IF(ABS(A-B)<EPSILON) THEN
   . . . .
ENDIF
```

şeklinde yapılmalıdır.

ÖRNEK 8: İkinci dereceden denklemin köklerini gerçek farklı, gerçek katlı kök ve sanal kökler olmak köklerinin türlerini belirten bir program yazınız.

```
PROGRAM ikinci_derece_denklem
REAL :: A, B, C, DELT
READ*,  A, B, C
DELT=B**2-4.0*A*C
IF(DELT>0.0) THEN
     ! DELTA>0 olduğu durumda kullanılacak deyimler Bu Blokta yer alır
     PRINT*, 'iki farklı gerçek kök var '
ELSE IF(ABS(DELT)<1.e-6) THEN
     ! DELTA=0 olduğu durumda kullanılacak deyimler Bu Blokta yer alır
     PRINT*, 'Eşit iki gerçek kök var '
ELSE
     ! Yukarıdaki her iki koşulun sağlanmadığı durum için (DELTA<0)
     ! kullanılacak deyimler bu blokta yer alır
    PRINT*,  'Kökler kompleks eşleniktir'
ENDIF
END PROGRAM ikinci_derece_denklem
```

Bu programda $\Delta = 0$ koşulunu vermek yerine $|\Delta| < 10^{-6}$ mantıksal ifadesi kullanılmıştır; çünkü sıfıra eşitlik koşulu koymak yukarıda bahsedilen nedenlerden ötürü yanlıştır.

ÖRNEK 9: a, b ve c sayılarının, bir üçgenin kenarlarını oluşturup oluşturmadığını saptayan bir program yazınız. Eğer bir üçgen oluşturuyorlarsa, üçgenin çevresini hesaplasın, aksi takdirde ekrana 'ÜÇGEN DEĞİLDİR' mesajını yazsın.

Bu problemin programını hazırlamak için gerekli bilgi, geometri dersinden hatırlayacağınız üzere "*üçgenin bir kenarı daima diğer iki kenarının toplamından küçük olmalıdır*" kuralıdır. Buna göre, $a < b + c$, $b < a + c$ ve $c < a + b$ olduğu göz önüne alınacaktır. Böylece a, b ve c sayıları $a \geq b + c$, $b \geq a + c$ veya $c \geq a + b$ eşitsizliklerinden en az birini sağlıyorsa, bir üçgenin kenarlarını oluşturamazlar.

```
PROGRAM ucgen
REAL :: A, B, C, CEVRE
PRINT*, 'A, B ve C yi Giriniz '
READ*, A, B, C
IF(A>=(B+C).OR.B>=(A+C).OR.C>=(A+B)) THEN
     PRINT*, 'ÜÇGEN DEĞİLDİR'
ELSE
     CEVRE=A+B+C
     PRINT*, 'Çevre=',CEVRE
END IF
END PROGRAM ucgen
```

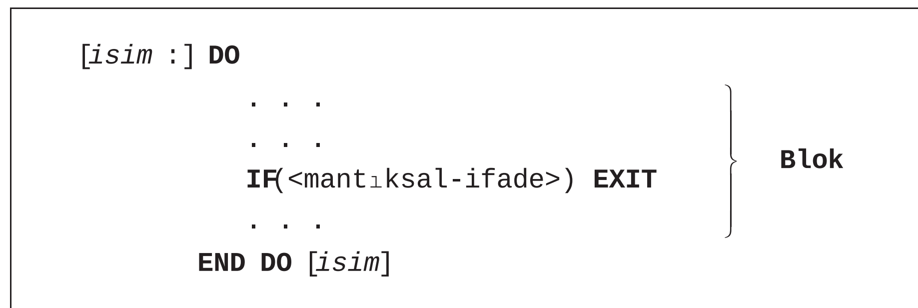
Yukarıda bahsedilen nedenle eşitsizliklerden herhangi biri sağlandığında, "*üçgen değildir*" mesajı yazdırılmıştır. Eşitsizlikler sağlanmıyorsa, yani IF deyimindeki mantıksal önerme FALSE değerini alıyorsa, verilen üç sayının bir üçgenin kenarlarını oluşturduğu kesinleşecektir. Bu koşul altında üçgenin çevresini üç kenarın toplanması ile buluruz.

ÖRNEK 10: Bir sigorta şirketinin uyguladığı sağlık sigortası primi esasları şöyle belirtilmektedir. Aylık prim (YTL olarak), şahıs yaşının 5 katı olarak belirleniyor. Ayrıca şahıs bekar ise (`Durum=1`) % 10, evli ve çocuksuz ise (`Durum=2`) % 15 ve evli ve çocuklu ise (`Durum=3`) % 25 iskonto uygulanıyor. Şahsın yaşını (`Yas`) ve durumunu (`Durum`) esas alarak aylık sigorta primini hesaplayan bir program yazınız.

```
PROGRAM Sigorta_Primi
INTEGER :: Durum, Yas
REAL    :: PRIM
PRINT*, 'ŞAHSIN YAŞINI GIRINIZ '
READ*, Yas
PRINT*, 'DURUM KODUNU GIRINIZ '
READ*, Durum
PRIM=5.0*Yas
SELECT CASE (Durum)
     CASE (1)
          PRIM=0.90*PRIM
     CASE (2)
          PRIM=0.85*PRIM
     CASE (3)
          PRIM=0.75*PRIM
     CASE DEFALUT   ! Durum<1 veya Durum>3
          EXIT
END SELECT
PRINT*, 'Şahsın Yaşı = ',Yas
PRINT*, 'Ödeyeceği prim = ',PRIM, ' YTL'
END PROGRAM Sigorta_Primi
```

3.6 DO İLE DO-WHILE DÖNGÜSÜ VE TEKRARLI ARİTMETİKLER

Bir `WHILE` döngüsü, bir mantıksal koşul sağlanıncaya kadar bir blok içindeki deyimlerin icra edilmesi işlemi için kullanılır. Genel kullanım şekli aşağıda verilmiştir:



`DO-END DO` blok deyimleri arasındaki deyimler, *`mantıksal-ifade`* doğru oluncaya kadar ve "doğru" olduğu durum da dahil olmak üzere blok içinde yer alan işlemler icra edilir. Bu bloktan **`EXIT`** deyimi ile çıkılır. Bir `WHILE` döngüsünden, birden fazla `IF` deyimi ile çıkış koşulları verilebilir.

Bir DO-WHILE döngüsünde, döngüden çıkış koşulu IF yapısı kullanmayı gerektirmez; döngü yapısında bir mantıksal ifade kullanarak aşağıdaki gibi belirtilir. Mantıksal ifade "*doğru*" ise blok içindeki deyimler icra edilir ve DO-WHILE deyimine döner; mantıksal ifade halen "*doğru*" olduğu sürece, blok deyimleri icra edilir. Bu işlemler mantıksal ifade yanlış oluncaya kadar sürdürülür; "*yanlış*" olduğunda END DO'dan sonraki ilk icra edilebilir deyime geçer.

```
[isim :] DO WHILE (<mantıksal - ifade>)
        Deyim_1        }
        Deyim_2        }
        . . .          } Blok
        Deyim_n        }
     END DO [isim]
```

Bu yapı daha genel WHILE döngüsünün, çıkış testinin deyimin başında yapıldığı, özel bir durumudur.



Daha genel programlamaya olanak sağladığı ve birden fazla koşul ile döngüden çıkışların olanaklı olduğu DO-END DO *döngüsünü,* DO WHILE-END DO *döngüsüne tercih ediniz!*

Programlama dillerinde öğrencilere ters gelen bir uygulama da, bilgisayarda yapılan çok sayıdaki toplama (veya çıkarma, çarpma, bölme) işlemi için izlenen metottur. Bu metotta değişkenlerin değerlerinin bellekte saklanması esasından yararlanır (*bknz* Kısım 2.6 ve 2.7).

ÖRNEK 11: Ekrandan girilen 10 adet sayının toplamını veren bir program yazınız.

Bu programın oluşturulabilmesi için Fortran dilinde mevcut birkaç yapının kullanılması ile mümkündür. Bu kısımda verilen yapıları kullanarak aşağıdaki şekilde sayaç kullanımını içeren iki program verilmiştir:

```
PROGRAM DoWhile_ornek
IMPLICIT NONE
REAL    :: toplam, sayi
INTEGER :: sayac
sayac=0
DO WHILE (sayac<10)
     sayac =sayac + 1
     PRINT*, sayac,"ci sayıyı giriniz"
     READ*, sayi
     toplam=toplam + sayi
END DO
PRINT*, "Toplam=",toplam
END PROGRAM DoWhile_ornek
```

veya

```
PROGRAM Do_ornek
IMPLICIT NONE
REAL    :: toplam, sayi
INTEGER :: sayac
sayac=0
DO
     sayac =sayac + 1          ! sayaç ← sayaç+1
     IF(sayac>10) EXIT
     PRINT*, sayac,"ci sayıyı giriniz"
     READ*, sayi
     toplam=toplam + sayi      ! Toplam ← Toplam+1
END DO
PRINT*, "Toplam=",toplam
END PROGRAM Do_ornek
```

olarak yazılabilir.

Her iki programda `sayac` sayıların kaç tane olduğunu takip eden bir sayaç, `toplam` ise tekrarlı işlemler sonunda 10 tane sayının toplamını verecek olan bir değişkendir. Bu programlarda kullanılan ifadeler matematiksel olarak, normalde, bir eşitsizliği göstermektedir: oysa daha önce bahsedilen aritmetik atama deyimlerine `sayac`←`sayac+`1 ve `toplam`← `toplam+x`'e karşılık gelmektedir.

Bu ifadeleri matematiksel olarak açıklamak için, rekürans bağıntılarını anlamak gerekir. Rekürans bağıntılı bir dizi ele elalım. Genel terimi $a_n = a_{n-1} + 3$ ve $a_0 = 0$ olsun. Bu dizinin $n > 1$ için elemanları şu şekilde bulunur: önce genel terimde *n*=1,2,… yazılarak

$$
\begin{aligned}
a_1 &= a_0 + 3 = (0) + 3 = 3 \\
a_2 &= a_1 + 3 = (3) + 3 = 6 \\
a_3 &= a_2 + 3 = (6) + 3 = 9 \\
a_4 &= a_3 + 3 = (9) + 3 = 12 \\
&\vdots \quad \vdots
\end{aligned}
$$

Bu şekilde dizinin elemanlarının sırasıyla 0,3,6,9,12.. şeklinde oluştuğu görülür. Dikkat edilirse $a_4 = 12$ olup dört kez üç'ün toplamına eşittir. Bilgisayardaki `sayac=sayac+`1 ifadesi işlem gördüğünde `sayac`'ın değerini bir öncekine göre bir artıran rekürans ifadesine benzetilebilir. Diğer yandan, `toplam=toplam + x` ise `toplam` değerine her işlemde `x` sayısını ekler. Bu ifadeler ilk işlem sırasında sağdaki `toplam` değeri ne ise onun üstüne ekleme yapar. Bu tür ardışık işlemde indisleri kaldırarak yazarsak, `toplam` = `toplam` + `x` ifadesi elde edilir ki, burada sağ tarafındaki toplam hesaplanır ve eşitliğin sol tarafındaki değişkenin belleğine atanmasına karşılık gelir.

Çok sayıda çarpma işlemi yapmak gerektiğinde de toplama işlemine benzer bir ardışık işlem uygulanabilir. Örneğin, *n!* hesabını yapan bir program hazırlayalım.

```
PROGRAM Faktoriyel
IMPLICIT NONE
INTEGER :: sayac=0, n, faktor=1.
READ*, n
     DO
        sayac =sayac + 1        ! sayaç ← sayaç+1
        IF(sayac>n) EXIT
        faktor=faktor*sayac     ! Faktör ← Sayaç* Faktör
     END DO
PRINT*, "Faktoriyel=",faktor
END PROGRAM Faktoriyel
```

Burada da `Faktor=Faktor*Sayac` ifadesi $Faktor_i = i * Faktor_{i-1}$ ve $Faktor_0 = 1$ şeklindeki bir rekürans bağıntısı, `Faktor←Faktor*Sayac`, şeklinde indisli değişken kullanımını gerektirmeyen atama komutu ile gerçekleştirilmektedir.

3.7 MAKİNE SABİTLERİ VE KULLANIMI

Programcılar uyguladıkları algoritmalar gereği kullanılan makinenin (bilgisayarın) işlem yapabileceği en küçük sayı (epsilon), sayının alt ve üst sınırları, en büyük sayı, basamak doğruluğu gibi özelliklerden yararlanırlar. Bu özellikler tek veya çift hassasiyet kullanımına göre değişmektedir. Kıyaslamalarda makinenin bu özelliklerinden yararlanılabilir.

FORTRAN 90/95 dillerinde, programda kullanılan basamak hassasiyetine göre, makinenin alabileceği en küçük, en büyük vb sayıları veren arşiv programları ile donatılmıştır. Bu arşiv programlarından en sık kullanılanları Tablo 3.3'de verilmiştir.

Tablo 3.3 Makinenin sayılarla ilgili arşiv programları ve işlevleri.

Fonksiyon	Kullanımı
`BIT_SIZE(i)`	Tamsayı sorgulama fonksiyonudur; `i` tamsayısının bit sayısını bildirir
`DIGITS(x)`	Tamsayı sorgulama fonksiyonudur; `x` gerçek veya tam sayısının değerli basamak sayısını ikilik düzende bildirir. Onluk düzendeki basamak sayısı için `PRECISION(x)` kullanılmalıdır.
`EPSILON(x)`	Tamsayı sorgulama fonksiyonudur; `x` gerçek sayısı ile aynı tipte olup makinenin hassasiyet bakımından en küçük sayısını bildirir.
`HUGE(x)`	Tamsayı sorgulama fonksiyonudur; `x` gerçek sayısı ile aynı tipte olup makinenin hassasiyet bakımından en büyük sayısını bildirir.
`KIND(x)`	Tamsayı sorgulama fonksiyonudur; `x` sabiti veya değişkeninin tipini bildirir.
`PRECISION(x)`	Tamsayı sorgulama fonksiyonudur; `x` sabiti veya değişkeninin değerli basamak sayısını onluk düzende bildirir.
`RANGE(x)`	Tamsayı sorgulama fonksiyonudur; `x` tipindeki (tamsayı, gerçek veya karmaşık sayı) sayının en küçük ve en büyük sayı aralığını bildirir.

ÖRNEK 12: Kişisel bilgisayarınızın, Tablo 3.3'de verilen arşiv programları kullanılarak, çeşitli sayısal kapasitesi sorgulayınız.

```
PROGRAM Ornek_12
IMPLICIT NONE
PRINT*, "Basamak   =",DIGITS(1.)
PRINT*, "Ufak      =",TINY(1.0)
PRINT*, "Ufak      =",TINY(1.d0)
PRINT*, "Epsilon   =",EPSILON(1.0)
PRINT*, "Epsilon   =",EPSILON(1.d0)
PRINT*, "Büyük     =",HUGE(1.0)
PRINT*, "Büyük     =",HUGE(1.d0)
PRINT*, "Hassasiyet=",PRECISION(1.)
PRINT*, "Hassasiyet=",PRECISION(1.d0)
END PROGRAM Ornek_12
```

Programının çıktısı

```
Basamak   = 24          ! Sayının ikilik düzende kullandığı basamak sayısıdır
Ufak      = 1.17549435E-38           ! en küçük sayı, REAL*4
Ufak      = 2.225073858507201E-308 ! en küçük sayı, REAL*8
Epsilon   = 1.19209290E-07           ! makine epsilonu REAL*4
Epsilon   = 2.220446049250313E-16  ! makine epsilonu REAL*8
Büyük     = 3.40282347E+38           ! en büyük sayı REAL*4
Büyük     = 1.797693134862316E+308 ! en büyük sayı REAL*8
Hassasiyet= 6                        ! basamak hassasiyeti REAL*4
Hassasiyet= 15                       ! basamak hassasiyeti REAL*8
```

olarak elde edilmiştir ve kullandığınız bilgisayarın mikro-işlemcisine göre farklılıklar gösterebilir. Kısım 3.5'de verilen `EPSILON` yerine `EPSILON(1.)` kullanılabilir.

ALIŞTIRMALAR

3.1 Aşağıdaki program parçalarında varsa hataları bulunuz.

```
(a)  REAL :: A
     IF(A==10) THEN
          Print*, "A=10 dur "
     ENDIF
(b)  REAL :: X
     IF(X>5.5) THEN
        X=5.5
     ELSE
        X=5.5
     ENDIF
(c)  REAL :: X
     INTEGER :: I
     IF((I>15).AND.(X<2.5)) THEN
          X=0.25*X
```

```
     ELSE IF((I<=15).AND.(X>=2.5)) THEN
          X=4.*X
          I=I-1
     ELSE
          Print*, "x=",x," i=",I
     ENDIF
(d)  INTEGER :: I, N
     IF(I==) THEN
         N=I+5
     ENDIF
(e)  REAL :: A, B, X
     IF(A+2.*B<=3.55) X==1.234

(f)  REAL :: A, G
     IF G>=3. THEN
       A=0.5*G*G
     END IF
(g)  REAL :: X, Y, A, B, C
     IF(X*Y) C=SQRT(A+1.25*B)
```

3.2 `L1`, `L2` ve `L3` mantıksal sabitlerin `L1=.TRUE.`, `L2=.TRUE.` ve `L3=.FALSE.` olarak değer atandığını kabul ederek aşağıdaki mantıksal işlemlerin sonuçlarını bulunuz.

```
(a) .NOT.L1              (b) L1.OR.L3
(b) L1.AND.L3            (d) L2.NEQV.L3
(e) L1.AND.L2.OR.L3      (f) L1.OR.L2.AND.L3
(g) .NOT.(L1.EQV.L2)     (h) (L3.AND.L2).OR.L1
```

3.3 Aşağıdaki işlemi yapan bir FORTRAN programları yazınız.

$$x_i = i^2 / 2^{i+1},\ y_i = 2^{i+3} / (1 + 3^{i+2}),\qquad \frac{\left[\sum_{i=1}^{9} x_i^2 - \sum_{j=1}^{10} (x_j y_j - 1)^2\right]^{1/4}}{\sum_{i=1}^{6} (x_i^2 + y_i^2)}$$

3.4 Bir üçgenin kenar uzunlukları `a, b, c` olarak veriliyor. Bu üçgenin bir dik üçgen olup olmadığını tespit eden bir program yazınız.

3.5 $y = 3t^2 e^{-0.025t} \sin^2(2t + \pi)$ denkleminin, t'nin 1 ile 10 arasındaki değerlerine karşılık gelen y değerlerini tablo halinde veren bir program yazınız. <u>*Not:*</u> Programınızı Δt artırım miktarı dışarıdan seçenek olarak girilecek şekilde yazınız.

3.6 $\sin x$ ve $\cos x$ fonksiyonlarının 0-180 derece arasında aldığı değerleri tablo halinde veren bir program yazınız. <u>*Not:*</u> Programınızı Δt artırım miktarı dışarıdan seçenek olarak girilecek şekilde yazınız. Arşivdeki trigonometrik fonksiyonlarda açı raydan olmalıdır; bu nedenle derece olan açının radyana dönüştürülmesi gerektiğini unutmayınız.

3.7 Ekrandan girilen 75 tamsayıdan (a) kaç tanesi "tek" kaç tanesi "çift" olduğunu bulan ve

adetlerini ekrana veren bir program yazınız (b) “tek” olanlar ile “çift” olanların ayrı ayrı toplamlarını hesaplayan bir program yazınız (c) “tek” olanlar ile “çift” olanların aritmetik ortalamalarını hesaplayan bir program yazınız.

3.8 Bir arı kovanında 5800 arı mevcuttur. Arıların haftalık çoğalma oranı %2.3 olduğuna, her iki haftada % 1.98'i çeşitli nedenlerle öldüğüne göre arıların bir yıl için, kovandaki arı nüfusunun haftalık olarak değişimini veren bir program yazınız.

3.9 Suyun deniz seviyesinde kaynama noktası 100°C dır. Su deniz seviyesinden daha yükseklerde daha düşük sıcaklıklarda kaynamaktadır. Yükseklik *H* (metre) olarak alındığında *Kaynama_Noktasi* (°C) aşağıdaki bağıntı ile hesaplanabilir:

$$\texttt{kaynama_noktasi} = 100 - \frac{H}{170} \quad (°C)$$

Bir FORTRAN programı yazarak deniz seviyesinden 100’er metre aralıklarla 5000 metre yüksekliğe kadar kaynama noktası sıcaklığını veren bir tablo oluşturunuz.

3.10 Bir dershanede derslikler günde on saat kullanılacak şekilde düşünüldüğünde en fazla 5,000 öğrenciye eğitim verecek kapasiteye sahiptir. Dershaneye ilk yıl 2,896 öğrenci kayıt olmakla birlikte, öğrenci kayıt oranın her yıl % 5.5 artacağı düşünülmektedir. Bu esnada yıllara göre bu artışın sabit kaldığı varsayılırsa, dershanenin tam kapasiteye ulaşmak için kaç yıl gerektiğini hesaplayan bir program yazınız.

3.11 5'i kız (K) ve 7'si erkek (E) 12 kişilik bir sınıftaki öğrencilerin matematik ve fizik sınavlarından aldıkları notlar aşağıdaki tabloda verilmektedir. Bu sınıftaki öğrencilerden (a) Her iki dersten geçer not alan (50 ve üstü) kız öğrencilerin sayısını, (b) Her iki dersten başarısız olan erkek öğrencilerin sayısını, (c) Matematikten başarılı fakat fizikten başarısız olan erkek öğrencilerin sayısını veren bir program yazınız.

Öğrenci Kodu	1	2	3	4	5	6	7	8	9	10	11	12
Cinsiyet	E	K	K	E	E	E	K	E	K	K	E	E
Matematik	13	52	61	66	74	48	63	42	93	37	24	15
Fizik	92	77	87	43	62	32	28	73	52	59	37	47

3.12 Bir banka, aylık % 3.9 faizle, en fazla 2 yıla kadar kredi vermektedir. 50,000 YTL’den başlayarak 5,000 YTL’lik artışlarla 200,000 YTL’e kadar kredi alınması halinde müşterinin ödeyeceği aylık taksitleri veren bir tablo oluşturmak istenmektedir. Bu koşullara uygun bir program yazınız. Kullanılacak formül aşağıda verilmiştir:

$$Taksit = \frac{rP}{1-(1+r)^{-n}}$$

P kredi miktarı (YTL), *r* aylık faiz oranı (bu durum için 3.9/100) ve *n* kredi süresi (ay olarak) ifade edilmektedir.

3.13 Dört adet “gerçek” `A`, `B`, `C`, `D`, bir “tamsayı” `I` değerlerini okuyan ve aşağıdaki işlemleri yapan bir program yazınız.

Eğer `I=1` ise `X=A+B+C+D`
Eğer `I=2` ise `X=(A+B+C+D)/4.0`
Eğer `I=3` ise `X=A/B+C/D`

Eğer `I=4` ise `X=(A+B)/(C+D)`
Eğer `I=5` ise `X=A/B-C/D`
Eğer `I=0` ise Programı Terk et

3.14 Mehmet satın aldığı dört vitesli bir bisikletin kullanım kılavuzuna baktığında, bisikletin dişli oranlarının aşağıdaki şekilde verildiğini gözlüyor.

Vites	Dişli Oranı
1	3.3/1
2	2.1/1
3	1.6/1
4	1.1/1

Bisikletin hızının aşağıdaki formülle hesaplanabileceği göz önünde tutulduğunda bisikletin girilen vites değeri ve devir sayısı için hızını hesaplayan bir FORTRAN programı yazınız.

```
HIZ = 0.028 * Devir_sayisi/Disli_orani
```

Bu formül uyarınca, örneğin birinci viteste devir sayısı 3300 devir/dakika için bisiklet hızı $0.028 \times 3300 \times (1/3.3) = 28$ km/h bulunur.

3.15 Aşağıdaki matematiksel işlemleri yapan ve ekrana `XO` ve `Y` değerlerini yazan bir program yazınız.

$$XO = \frac{1}{8}\sum_{n=1}^{8}(x_n^2 + 1) \qquad x_n = \left(\frac{2}{3}\right)^n \qquad Y = \sqrt{\frac{1}{7}\sum_{n=1}^{8}\left(\frac{n}{XO} - \frac{1}{x_n}\right)^2}$$

3.16 İki bilinmeyenli iki denklem

$$A \cdot \mathbf{X} + B \cdot \mathbf{Y} = C, \qquad D \cdot \mathbf{X} + E \cdot \mathbf{Y} = F$$

olarak verilmiş olsun. Programa dışarıdan girilecek *A, B, C, D, E* ve *F* katsayıları için *X* ve *Y*'nin değerlerini hesaplayan bir FORTRAN programı yazınız.

3.17 Bir'den 20'e kadar sayıların bir birleriyle çarpımlarını bir "çarpım tablosu" şeklinde veren bir program yazınız.

3.18 Bir'den 100'e kadar olan tamsayıların 2'şer artırımlarla sırasıyla n, n^2, n^3, $\sqrt[3]{n}$ değerlerini tablo şeklinde veren bir FORTRAN programı yazınız.

3.19 Kütlesi m (kg) olan bir cisim, h (m) yüksekliğinden serbest düşmeye bırakıldığında, Δt (sn) aralıklarla, yere vuruncaya kadar, cismin konumunu, hızını ve enerjisini tablo halinde veren bir program yazınız. Program girdileri m, h ve Δt olarak alınacaktır.

3.20 Hava ile sürtünmenin ihmal edilebildiğini kabul edersek, yeryüzünden havaya u_0 (m/s) hızıyla ve yüzeyle θ (derece) açısı yapan bir açı ile fırlatıldığında, cisim parabolik bir yörünge çizer. Cismin yörünge denklemleri

$$x(t) = x_0 + (u_0 \cos\theta)t \quad \text{ve} \quad y(t) = y_0 + (u_0 \sin\theta)t + \frac{1}{2}gt^2$$

ile verilmektedir. Burada (x_0, y_0) cismin fırlatıldığı konumun koordinatlarını (m), t cismin uçuş anını (sn) ve g yerçekimi ivmesini (m/s^2) olarak verilmektedir. Cismin (0,0) konumundan 2, 5, 8, 11 ve 14 m/s hızlarıyla, 10, 25, 40, 55, 70 ve 85 derece açılarla fırlatıldığında cismin menzilini (cisim yere vurduğundaki x değeri) ve uçuş süresini, döngüler kullanarak hesaplayan bir program yazınız. *Uyarı:* Trigonometrik fonksiyonlar radyan cinsinden açılarla işlem yapar bu nedenle derece olan açıları $\theta_{rad} = \pi\theta_{derece}/180$ ile radyan birimine dönüştürünüz.

3.21 Dört cismin kütleleri 4.5, 12, 18.5 ve 27.3 kg olarak verilmiştir. Her cisim sırasıyla 3, 6.5, 8.2, 12 ve 16.1 m/s sabit hızla hareket ettirilmek istense, bütün cisimlerin farklı hızlardaki kinetik enerjisini hesaplayan bir program hazırlayınız. *İpucu:* Tüm işlemi tek program içinde yapabilmek için döngü kullanımından yararlanınız.

3.22 Kurt ile tavşan nüfusunun ekolojik simülasyonu yapılmak isteniyor. Simülasyonun yapıldığı yörede tavşanlar yeşil, kurtlarda sadece tavşan ile beslenmektedir. Yörede çok sayıda yeşillik var iken tavşanlar ile beslenen kurtlar bir dezavantaja sahiptir. Kurt nüfusu yiyecek miktarı (tavşan) ile artmaktadır. Tavşan ve kurt nüfusunun günlük değişimi (sırasıyla T ve K) aşağıdaki formüller ile verilmektedir.

`Tavsan` (yarın) `= (l+a-c*K)*Tavsan` (bugün)
`Kurt` (yarın) `= (l-b+c*d*T)*Kurt` (bugün)

Burada `a` tavşanlar kurtlar tarafından avlanmadıkları takdirde çoğalma oranı 0.01 (%1), `b` yiyecek tavşan bulamamaları takdirde kurt nüfusundaki azalma oranı 0.005 (%0.5), `c` bir kurt 'un bir tavşan ile karşılaşma ve tavşanı yeme ihtimali 0.00001 (% 0.001) ve `d` tavşan ile beslenen kurtların nüfusundaki artış oranı 0.01 (%1) olarak alınmaktadır.

Kurtlar ile tavşanların günlük nüfus seviyelerini 1 yıl için (365 gün) hesaplayan bir program yazınız. Başlangıçta 10,000 tavşan ve 800 kurt mevcut olduğunu kabul ediniz. Daha sonra kurt sayısını iki misli artırarak hesabı yenileyiniz.

3.23 Bir basit AB kirişinin herhangi bir P noktasına yerleştirilen F tekil yükü sonucunda A ve B mesnetindeki reaksiyonlar aşağıda verilmiştir:

$$R_A = \frac{Fb}{\ell}, \qquad R_B = \frac{Fa}{\ell}$$

Eğilme momenti ve X mesafesindeki sehim (yer değiştirme) aşağıdaki denklemler ile hesaplanabilmektedir:

$$X \le a, \qquad (BM)_x = \frac{FbX}{\ell}, \qquad (DE)_x = -\frac{FbX(\ell^2 - b^2 - X^2)}{6EI\ell}$$

$$X > a, \qquad (BM)_x = \frac{Fa(\ell - X)}{\ell}, \qquad (DE)_x = -Fa(\ell - X)\frac{(\ell^2 - b^2 - (\ell - X)^2)}{6EI\ell}$$

Burada $(BM)_x$, X'deki eğilme momenti (kg.m), $(DE)_x$, X'deki sehim (m), E, elastisite modülü (kg/m^2), I=atalet momenti (m^4), $\ell = |AB|$ mesafesi (m), $a = |AP|$ mesafesi $b = |PB|$ mesafesidir. Maksimum eğilme momenti

$$(BM)_{max} = \frac{Fab}{\ell}$$

ve maksimum sehim
$a > b$ b için A noktasından itibaren

$$X = \sqrt{\frac{a(\ell + b)}{3}} \quad \text{ve} \quad (DE)_{\max} = -\frac{FbX^3}{3EI\ell}$$

$b > a$ için B noktasından itibaren

$$X = \sqrt{\frac{b(\ell + a)}{3}} \quad \text{ve} \quad (DE)_{\max} = -\frac{FaX^3}{3EI\ell}$$

Buradaki *X*, artık maksimum sehimin oluştuğu yerdir. Reaksiyonları, eğilme momentlerini ve sehimi $\Delta\ell$ aralıklarla hesaplayan bir program yazınız. Yazacağınız programda kiriş özellikleri (*E* ve *I*) ile uzunluğu (ℓ), aralık miktarı ($\Delta\ell$), yükün yerleştirildiği nokta (*P*) ve yük değişken olarak seçilebilmelidir.

3.24 Bilgisayarlı hava sıcaklığı tahmini yapmak için izlenen metotlardan birisi şu şekilde verilmektedir: ölçüm istasyonundan itibaren, rüzgarın estiği yöne göre, her 10 km/h için sıcaklık düşüşleri kullanılıyor. Örneğin, herhangi bir istasyon için ve herhangi bir anda, rüzgarın yönü doğu, kuzey, batı ve güney yönlerindeki sıcaklık düşüşleri sırasıyla 1.08°C/10 km/h, 0.87°C/10 km/h, 0.64°C/10 km/h ve 0.53°C/10 km/h olmaktadır. Bu anda, istasyon ortam sıcaklığı, rüzgar hızı (km/h) ve yönü girildiğinde, hava sıcaklığını hesaplayan bir program yazınız. NOT: Rüzgar yönü Kuzeydoğu ise sıcaklık düşüşü Kuzey ve Doğu için verilen değerlerin ortalaması olarak alınacaktır. Ayrıca, yönleri belirlemek için yön kodu (örneğin, `Yon=1` yani Kuzey, `Yon=2` yani Kuzey Doğu, `Yon=3` yani Doğu vb) kullanınız.

3.25 Bir araba kiralama firması çeşitli model arabalarını günlük ve haftalık olarak kiralamaktadır. Bir arabanın kira bedeli, kiralanan gün sayısı (`kira_gun`) çarpı günlük kira bedeli (`Gunluk_Bedel`) artı aracın aldığı yol (`Yol`) çarpı kilometre başına bedel (`km_bedel`) ile hesaplanıyor. Haftalık kiralama durumunda ise, haftalık kira bedeli (`Haftalik_Bedel`) artı kullanılan benzin ücretidir. Bu durumda haftalık kira bedeli, gidilen yol, benzin sarfiyatı `Sarfiyat` (km/lt) ve benzin fiyatının `BenzinFiyat` (YTL/litre) bir fonksiyonudur. Firmanın elindeki arabalar normal, spor, lüks limuzin olarak üç gurupta toplanıyor. Bu guruplara ait veriler aşağıdaki tabloda özetlenmektedir.

	Günlük Bedel (*YTL/gün*)	**Kilometre Bedeli** (*YTL/km*)	**Haftalık Bedel** (*YTL/hafta*)	**Benzin Sarfiyatı** (*km/lt*)	**Benzin Fiyatı** (*YTL/lt*)
Normal	40	0.52	230	12	2.52
Spor	60	0.94	370	16	2.68
Limuzin	110	1.15	700	13	2.76

Araba kiralamak isteyen bir şahıs, günlük veya haftalık bazda, kiralayacağı bir araca ne kadar kira bedeli ödeyeceğini veren bir program yazınız. NOT: Haftalık kira bedelini hesaplarken, bir hafta boyunca araba ile katedilen yolun girilmesi gerekir.

BÖLÜM 4
KÜTÜK AÇMA/KAPAMA VE KULLANMA İŞLEMLERİ

Bir programın girdi verileri az sayıda ise, verilerin bir kısmı ekrandan, bir kısmı program içinde sabit atama deyimleri ile girilebilir. Ancak veri sayısı fazla ise, bunları programı her çalıştırmada ekrandan girmek pratik bir yol değildir. Binler ve on binlerle ifade edilen verileri de program içinde tanımlamak iyi bir programlama tekniği değildir. Verilerin, hem programlamanın pratik ve genel olması hem de bazı uygulamalara daha uygun olması açısından, bir veya daha fazla kütükten okunması tüm programlama dillerinde bir alternatif seçenek olarak sunulmuştur.

Çok sayıda veri kütükleri de genelde bilgisayar programları ile hazırlanır; örneğin, ÖSS sınavına girmek için başvuran adaylar, kişisel bilgilerin (TC Kimlik numarası, adı, soyadı, doğum tarihi, adresi, mezun olduğu okul ve türü, vb) girilmesinin istendiği bir optik form doldurur. Bu formlar, bir optik okuyucudan geçirildiğinde, sayısal ve alfa sayısal bilgilere dönüştürülerek bir kütüğe kaydedilir. Bu kütükte iki milyona yakın öğrenciye ait yaklaşık 20 kişisel bilgi (toplam kırk milyon veri) mevcut olacaktır. Bu bilgi daha sonra çeşitli amaçlara hizmet eden programlarda kullanılmak üzere saklanır. Bu tür veri kütüklerine *Veri Tabanı* adı verilir. Bir çok özel ve tüzel şirket, kendisi için önemli ve gerekli bilgilerin yer aldığı, veri tabanını oluşturur.

Kütüklere verileri kaydetmek veya kütükteki kayıtları programlara okutmak için bazı özel deyimler mevcuttur. FORTRAN 90/95 dilinde girdi/çıktı (**I/O**=**I**nput/**O**utput) deyimleri özet olarak Tablo 4.1'de verilmiştir. Bu deyimler kütük açma/kapama işlemlerine, kütüğün statüsünü kontrol etmeye, okuma imlecini kütük içinde belirli bir konuma götürme, kütükten okumaya ve/veya yazmaya olanak sağlar.

Tablo 4.1 FORTRAN 90/95 dilinde kullanılan I/O deyimleri.

Deyim	*İşlevi*
OPEN	Bir I/O cihazına bağlı kütük açma
CLOSE	Bir I/O cihazına bağlı kütüğü kapama
INQUIRE	Bir kütüğün özelliklerini araştırma
READ	Bir kütükten veri okuma
PRINT	Standard çıktı cihazına veriler yazdırma
WRITE	Bir kütüğe veri yazdırma
REWIND	Sıralı erişimli bir kütüğün okunmasını başa alma
BACKSPACE	Sıralı erişimli bir kütükte bir kayıt geriye gitme
ENDFILE	Sıralı erişimli bir kütüğün sonuna gitme

Bu deyimlerin işlevlerinin ayrıntısına girelim.

4.1 OPEN DEYİMİ

FORTRAN'da program kütüklerini yazmaya ve/veya okumaya açma deyimi **OPEN** deyimidir. Daha önce bahsedildiği gibi, bir bilgisayarın iki tür belleği vardır: *ana bellek* (daha hızlı) ve *ikincil bellek* (daha yavaş). Veri kütükleri daima ikincil bellekte, yani bir disk, CD veya teypte depolanır. Fazla sayıda veri gerektiren veya üreten programlarda bu verilerin giriş/çıkışının kütükler ile sağlanması daha pratiktir.

Veriler kütüğe yazıldıktan sonra, kütük sonu *end of file*—EOF ile işaretlenir. Bu kütüklere *Sıralı Erişimli Kütük* (SEK) adı verilir. Bu kütüklerdeki bilgilere, kütüğe yazıldıkları sırayla erişilebilir. Bu işlem binlerce ve hatta milyonlarca veri kullanıldığında oldukça verimsiz bir durum veri okuma arz eder. Manyetik diskin özelliklerinden faydalanan FORTRAN programlama dili, ikinci tip bir kütük kullanımını mümkün kılar.

Manyetik disk bilgiyi mıknatıslanmış *bit*ler olarak daha sıkı bir şekilde depolar. Bilgi bir erişim kolu ile diskten okunur. Diske kaydedilen bilgi her satırına bir sıra kayıt numarası atanan satırlara rasgele yerleştirilebilir. Bu tür kütüklere de *Rastgele (Doğrudan) Erişimli Kütükler* (REK) adı verilir.

4.1.1 OPEN DEYİMİNİN KULLANIMI

OPEN deyimi icra edilen bir Fortran deyimidir. Programa bir kütük atama işlemi için OPEN deyimi çeşitli özelliklerde kullanılır. Bir disk kütüğünün bazı özelliklerinin OPEN deyiminde belirlenmesi aşağıda verilmektedir:

1. Bir kütüğün cihaz birim numarası UNIT tanımlanmalıdır. Örneğin, cihaz numarası UNIT=10 olan bir girdi kütüğünden veriler READ(10,..) deyimi ile okunur. UNIT=12 olarak tanımlanan bir çıktı kütüğünde kullanılan WRITE deyimleri WRITE(12,..) şeklinde olduğunda çıktılar bu kütüğe kaydedilir (Standard olarak girdilerin ekran veya klavyeden girilmesinde cihaz numarası 5, çıktıların ekrana verilmesinde 6'dır).
2. Kütüğün kullanım türü (*girdi/okuma* veya *çıktı/yazma* kütüğü mü?) belirtilir.
3. Girdi veya çıktı kütüğün diskte kayıtlı olduğu adı tanımlanır.
4. Sıradan veya rasgele erişimli kütük olarak mı hazırlanmış olduğu belirtilir.
5. Boşlukların boşluk ya da sıfır olarak mı değerlendirileceği belirtilebilir.

OPEN deyiminin genel şekli aşağıda verilmiştir:

```
OPEN (UNIT=u, ERR=sl,FILE=fname,STATUS=stat,FORM=form, &
     IOSTAT=io, ACCESS=acc, BLANK=blnk, ACTION=action, &
     POSITION=konum, PAD=pad, RECL=kay)
```

- **`UNIT`**=*u* veya sadece *u* şeklinde kullanılabilir. Burada u bir tamsayı olup açılacak kütüğün *cihaz* veya *birim* numarası olarak adlandırılır. Bir programa okuma (girdi) veya yazma (çıktı) amacıyla açılan her kütüğün bir numarası olması gerekir. `OPEN` deyiminde kullanılması *zorunlu* olan tek belirteçtir. `UNIT` anahtar kelimesinin kullanılması zorunlu değildir; ancak bu durumda *u* belirteci `OPEN` deyiminin ilk belirteci olarak kullanılmalıdır. `UNIT` anahtar kelimesiyle beraber kullanıldığında belirteçlerin içindeki sırası önemli değildir.

- **`FILE`**=*kütük_ismi*. Otuzbir veya daha az karakterden oluşan, ilk karakteri bir harften ibaret olan kütük ismidir. Kütük ismi, tek veya çift tırnak işaretleri arasında verilmelidir; yani bir alfa sayısal sabit olarak girilmelidir.

- **`STATUS`**=*stat*. ilgili kütüğün dört durumunu (statüsünü) bildiren alfa sayısal bir sabittir.
    - `STATUS='`**`OLD`**`'` olarak tanımlandığında kütüğün diskte var olduğunu, yani eski kütük olduğunu, ve veri okuma amacıyla kullanılacağını belirtir. Dolayısıyla bu kütüğün önceden hazırlanmış olması gerekir.
    - `STATUS='`**`NEW`**`'` durumunda çıktı yazdırmak amacıyla yeni bir kütük oluşturulacağını bildirir.
    - `STATUS='`**`REPLACE`**`'` durumunda kütük mevcut olsun veya olmasın yeni bir kütük açılır. Kütük mevcutsa, yazılan veriler, eski bilgilerin silinmesine neden olur.
    - `STATUS='`**`UNKNOWN`**`'` ile kütüğün statüsünün belirsiz olduğunu (yani hem eski (girdi) hem de yeni (çıktı) kütüğü olabilir anlamında) bildirir.
    - `STATUS='`**`SCRATCH`**`'` seçeneği çok sayıda veri üreten programlarda, programcı ana belleği (RAMi) kullanmaktan veya işgal etmekten ziyade, bunları yardımcı belleğe, yani diske, geçici bir süreliğine boşaltarak; bu bilgilere ihtiyacı olduğunda diskten okuma yoluna başvurur. Bu verilerin kullanımına ihtiyaç kalmadığında, verilerin yer aldığı kütük(ler) silinir. Yani bu seçenek ile açılan kütük, hem çıktı hem de girdi kütüğü gibi işlev görür ve program terk edilirken diskten silinir.

- **`IOSTAT`**=*io_durumu*, girdi/çıktı belirteci bir tamsayı olup, hata saptandığında pozitif bir tam değer aksi halde *sıfır* değerini almaktadır.

- **`ACCESS`**=*acc*. Kütüğün SEK mi yoksa REK mi olarak açılacağını belirtir, *acc*'nin aldığı geçerli seçenekler SEK için `'`**`SEQUENTIAL`**`'` ve REK için `'`**`DIRECT`**`'` şeklindedir.

- **`FORM`**=*form* seçeneği ile açılan kütüklere `'`**`FORMATTED`**`'` ve `'`**`UNFORMATTED`**`'` değerleri atanabilir; kütüğe kaydedilen değerlerin *formatlı* veya *formatsız* olacağını belirtir. `ACCESS='DIRECT'` seçeneği ile beraber *form* belirtilmemişse `'UNFORMATTED'`, `ACCESS='SEQUENTIAL'` ise ve form belirtilmemişse `'FORMATTED'` olarak işlem görür.

- **`RECL`**=*kay*, kaydın (formatlı veya formatsız) sabit uzunluğunu tanımlar; sadece doğrudan erişimli kütüklerde kullanılır; yani, seçenek `ACCESS='DIRECT'` ile kullanılır. Örneğin, `RECL=80` ifadesinde bir satırın maksimum uzunluğunun formatlı kayıtlar için 80 karakter, formatsız kayıtlar için 90 bayt olduğu anlaşılır.

- **`BLANK`**=*blnk*, bir karakter ifadesi olup `'`**`ZERO`**`'` ile boşlukların sıfır ve `'`**`NULL`**`'` ile boşluk olarak algılanmasına yarar. Bu seçenek kullanılmazsa, karakterler arası boşluklar "boşluk" olarak kabul edilir.

- **`ACTION=`***action*, skalar otomatik `CHARACTER` ifadedir. Dosyanın sadece girdi amacıyla

bağlanması halinde “okuma” (ACTION='**READ**'), sadece çıktı olarak açılması halinde “yazma” (ACTION='**WRITE**', hem okuma hem de yazma durumunda ACTION= '**READWRITE**' seçeneğini alır.

- **POSITION**=*konum*, kütük açıldıktan sonra, diskten kütük bilgisi okuyan ucun konumunu belirtir. Konum için kullanılabilen seçenekler, '**REWIND**', '**APPEND**' ve '**ASIS**'dir. REWIND seçeneğinde okuyucu kütük başındaki ilk kayda getirilir; APPEND seçeneğinde okuyucu kütükteki en son kayıtın sonuna; ve ASIS ile kütük okuyucu işlemci bağımlıdır. Bu deyim kullanılmadığında otomatik ayar seçeneği ASIS’tir.
- **PAD**=*pad*, formatlı girdi kayıtlarının boşluklar içerip içermediğini ('**YES**', '**NO**') belirtir. Otomatik seçenek değeri PAD='YES'tir.

Örneğin,

```
OPEN(UNIT=9,FILE='VERI.DAT')
```

veya

```
OPEN(9,FILE='VERI.DAT')
```

şeklinde açılan birim numarası 9 olan VERI.DAT kütüğü

```
OPEN(UNIT=9,FILE='VERI.DAT',STATUS='UNKNOWN', &
     ACCESS='SEQUENTIAL',BLANK='NULL')
```

ile eş anlamlıdır.

```
OPEN(22,STATUS='SCRATCH')
```

Örneğinde birim numarası 22 olan kütük açılır. Sistem bu kütüğe otomatik olarak bir isim verir. Kütük, program içinde CLOSE deyimi ile kapatılırsa veya program sonuna (END PROGRAM deyimine) ulaşıldığında otomatik olarak silinir.

```
OPEN(UNIT=9,FILE='a.dat',STATUS='OLD', &
    POSITION='APPEND',ACTION='WRITE')
```

Örneğinde birim numarası 9 olan a.dat isimli kütük açılır. Bu kütük statüsü OLD olduğundan bilgisayarda (diskette) mevcuttur ve bu kütükten veri okunmasına müsaittir. Kütük statüsü APPEND olduğundan, kütük okuyucu son kayıttan sonraki ve EOF işaretinden hemen önceki konumdadır. Kütük sadece yazmaya ayarlı, formatlı, sıralı erişimli bir kütüktür.

4.1.2 IOSTAT SEÇENEĞİNİ KULLANMANIN ÖNEMİ

Bir kütük açma işlemi başarısızlıkla sonuçlanırsa ve programın OPEN deyiminde IOSTAT seçeneğine yer verilmemişse, program bir hata mesajı verir ve çalışmayı durdurur. Büyük hacimli ve çalıştırması uzun süren programlarda, bu durum bir çok işlem yapıldıktan sonra verilerin kaybolması anlamına gelebilir. Bu nedenle, hatayı izole etmek bakımından OPEN deyiminde IOSTAT seçeneğine yer verilmelidir. Böylece kullanıcı programın çalışmaya devam etmesini isteyebilir veya uygun bir şekilde sonlandırılmasına olanak verir.

Genellikle IOSTAT seçeneğinin kullanılması, programcıya daha fazla esneklik sağlar.

```
INTEGER :: istat
OPEN(UNIT=8,FILE='TEST.DAT',STATUS='OLD',IOSTAT=istat)
!   Hata kontrolü yap…
IF(istat/=0) THEN
     PRINT*, 'Kütük açma işlemi başarısız oldu! '
     PRINT*, 'IOSTAT=',istat
     . . .
END IF
```



`OPEN` *ile kütük açma işleminde daima* `IOSTAT` *seçeneğini kullanınız!*

4.2 REWIND VE CLOSE DEYİMLERİ

Bir veri kütüğü kullanırken, bazen bilgisayarın kütük açıldıktan sonra, okuma ucunun (bir iğne şeklinde olan bu uç manyetik okuma yapar) kütüğün başına konumlandırılması istenir. Bu işlem `REWIND` deyimi ile yapılır. Genel kullanım şekli,

```
REWIND u
REWIND (u)
REWIND (UNIT=u, IOSTAT=istat)
```

olarak verilmektedir. Burada kullanılan anahtar kelime ve belirteçlerin tanımı daha önce `OPEN` deyimindeki tanımlamalarla uyumlu olmalıdır. `u` kütüğün cihaz numarasıdır.

Bir diğer programlama ihtiyacı da, kütüğün program ile ilişkisinin `CLOSE` deyimi ile kesilmesidir. Kütük kapatma işlemi

```
CLOSE u
CLOSE (u)
CLOSE(UNIT=u, STATUS=statü, IOSTAT=istat)
```

şeklinde uygulanır. `u` kütüğün cihaz numarası olup bu deyimdeki `UNIT`, `STATUS` ve `IOSTAT` belirteçleri `OPEN` deyiminde verilenlerle aynı işleve sahiptir. Ancak, statü `'KEEP'` ve `'DELETE'` seçenekleri alabilen bir alfa sayısaldır. Program sonuna ulaşıldığında, açılan kütüğün *korunması* veya *silinmesi* amacıyla kullanılır. Örnekler:

```
OPEN(7,FILE='A.DAT',STATUS='OLD')
. . .
READ(7,*) X,Y,Z        ! Programdaki ilk READ deyimi
REWIND 7
. . .
READ(7,*) A,B,C        ! REWIND komutundan sonraki ilk READ deyimi
```

Burada `REWIND` deyimi kullanıldıktan sonraki ilk `READ` deyiminde `A=X`, `B=Y` ve `C=Z` olmaktadır.

Aşağıdaki örnekte, `test.dat'` isimli yazdırma amaçlı "yeni" bir kütük açılmıştır. Yazdırma işlemleri bittikten sonra `CLOSE` deyimi ile yazdırmaya karşı kapatılmıştır; yani bu kütüğe yeni bir veri yazdırılmak istenirse kütüğün tekrar `OPEN` deyimi ile yazmaya açılması gerekecektir.

```
OPEN(15,FILE='test.dat',STATUS='NEW')
. . .
WRITE(15,*)  A1,B,CD,I3
. . .
CLOSE(UNIT=15)
```

Aşağıdaki örnekte daha önce açılmış olan 12 numaralı kütük bu deyim ile silinmektedir. Silme işleminin başarılı olup olmadığını anlamak bakımından kütük statüsü kullanılabilir.

```
CLOSE(UNIT=12,STATUS='DELETE',IOSTAT=istat)
```

ÖRNEK 1: `A.DAT` kütüğünden veri okuyan, aşağıda verilen Fortran programının çıktısı ne olur? `A.DAT` kütüğünün içeriği aşağıdaki gibi verilmektedir:

```
1.     -5.     3.
2.      7.     5.
```

```
PROGRAM Ornek1
REAL :: A, B, X, Z, U, V, Y
OPEN(UNIT=8, FILE='A.DAT',STATUS='OLD')
READ(8,*) A, X, Z
READ(8,*) B
PRINT*,  A, X, Z, B
REWIND 8
READ(8,*) U, V
Y=U+V
PRINT*, U, V, Y
END PROGRAM Ornek1
```

İlk okuma satırında okunan `A`, `X` ve `Z` değişkenlerine, `A.DAT` isimli kütükte `1.`, `-5.` ve `3` değerleri karşılık gelmektedir. Bundan sonra okunan değer olan `B`'ye `2.` karşılık gelir. `REWIND` deyimi ile okuyucu kol `7.` sayısının başında iken kütük başına getirilir (yani `1.`'in başına) ve bundan sonra okunan değerler `U` ve `V`'ye karşılık gelen değerler sırasıyla `l.` ve `-5.` olur. `Y=U+V=l.+(-5.)=-4.` olarak hesaplanır. Çıktı da dolayısıyla aşağıdaki şekli alır.

```
1.00000   -5.00000  3.00000   2.00000
1.00000   -5.00000  -4.00000
```

Aşağıdaki örnekte 10 birim numaralı ve `'VERI'` isimli kütüğün 128 bayt sabit uzunluğundan oluşan doğrudan erişimli bir disk kütüğünü tanımlamaktadır.

```
OPEN(10, FILE='VERI', RECL=128, ACCESS='DIRECT')
```

Kütüklerin kullanımlarına ve tiplerine ilişkin karakteristikler Tablo 4.2'de özetlenmiştir.

4.3 BACKSPACE DEYİMİ

Cihaz numarası *u* ile verilen kütükte okuyucu uç *m* kaydında bulunuyorsa, bu deyimin kullanımı ile göstergenin (*m*-l).ci kayıda gerilemesine, yani bir kayıt geriye gitmesine, neden olur. Bu deyimin genel kullanım şekli.

```
BACKSPACE u
BACKSPACE (u)
BACKSPACE(UNIT=u, STATUS=statü, IOSTAT=istat)
```

şeklinde kullanılır. Buradaki `UNIT`, `STATUS` ve `IOSTAT` belirteçleri `OPEN` deyiminde verilenlerle aynı işleve sahiptir.

Aşağıdaki örneklerde 8 numaralı kütükte okuma kaydı bir kayıt geriye (bir kayıt genelde satırdan oluşur) gider. Cihaz/kütük numarası 10 olan kütükte geriye gitme işleminde başarısızlık olup olmadığı `IOSTAT` ile kontrol edilmektedir.

```
BACKSPACE 8
. . .
BACKSPACE (UNIT=10, IOSTAT=istat)
```

4.4 ENDFILE DEYİMİ

Bu deyim sıralı erişimli bir kütükte gelinen konumunda EOF, yani kütük sonu, işaretini koyar ve kütüğün kapatılmasını sağlar. Bir kütükte `ENDFILE` deyimi icra edildikten sonra ne `READ` ne de `WRITE` işlemleri yapılabilir; bu işlemleri yapabilmek için `REWIND` veya `BACKSPACE` komutlarının kullanılması gereklidir. Genel kullanım şekli

```
ENDFILE u
ENDFILE (u)
ENDFILE (UNIT=u, IOSTAT=istat)
```

olarak verilmektedir. Bu deyim ile çıktı kütüğüne EOF işareti konur. Sadece sıralı erişimli kütüklerde uygulanabilir.

4.5 INQUIRE DEYİMİ

Bir kütük veya giriş/çıkış birimi hakkında kullanıcıya bilgi sağlar. Bu bilgiler; kütüğün mevcut olup olmadığını, hangi cihaza bağlı olduğunu, verilere erişimin doğrudan mı yoksa rasgele mi oluşu, doğrudan erişimli ise kayıt uzunluğu ve numarası, formatlı veya formatsız oluşu, veriler arasındaki boşlukların program tarafından yorumlanma şekli v.b olmaktadır. Bu deyim, kütük bir birime bağlanmadan, bağlanırken veya bağlandıktan sonra kullanılabilir. Genel kullanım şekli;

```
INQUIRE ( FILE=kütük-ismi,IOSTAT=I/0-durumu, EXIST=mevcut, &
  OPENED=açıkmı,NUMBER=sayı, NAMED=isimlimi,NAME=isim,     &
  SEQUENTIAL=sek,DIRECT=dek,FORMATTED=formatlı,            &
  UNFORMATTED=formatsız,ACCESS=erişim,FORM=şekil,          &
  RECL=kayıt_uzunluk,NEXTREC=sonraki-kayıt,BLANK=boşluk,   &
  ACTION=action,READ=oku,WRITE=yaz,READWRITE=okuyaz,       &
  DELIM=karakter,PAD=pad,IOLENGHT=uzunluk)
```

olarak verilmektedir. Buradaki parametrelerin daha detaylı olarak açıklamaları aşağıda verilmektedir:

- **FILE=kütük-ismi**, FILE anahtar kelimesinin kullanılması zorunludur. Araştırılan kütüğün ismi girilir.
- **IOSTAT=I/O-durumu**, girdi/çıktı belirteci bir tamsayı olup hata saptandığında pozitif aksi halde sıfır değerini alır.
- **EXIST=mevcut**, mevcut bir mantıksal değişkendir. Belirtilen kütüğün bulunamaması durumunda .FALSE. aksi halde .TRUE. değerini alır.
- **OPENED=açıkmı**, açıkmı belirteci mantıksal bir değişkendir. İsmi belirtilen kütük açılmış ise .TRUE. değilse .FALSE. değerini alır.
- **NAMED=isimlimi**, isimlimi soruşturulan kütüğün ismi varsa .TRUE. yoksa .FALSE. değerini alan bir mantıksal değişkendir.
- **NAME=isim**, kütüğün bir ismi bulunması durumunda ismin atandığı bir alfa sayısal değişkendir.
- **SEQUENTIAL=sek**, *sek* bir alfasayısal değişken olup erişimin sıralı olup olmadığını bildirir. Sıralı ise **'YES'**, değilse **'NO'** veya bir durum bildirmiyorsa **'UNKNOWN'** değerini alır.
- **DIRECT=rek** , *rek* bir alfasayısal değişken olup erişimin rastgele olup olmadığını bildirir. Rasgele ise 'YES', değilse **'NO'** veya bir durum bildirmiyorsa **'UNKNOWN'** değerini alır.
- **FORMATTED=formatlı** , *formatlı* bir alfasayısal değişken olup, kütüğün formatlı olup olmadığını bildirir. Formalı ise **'YES'**, değilse **'NO'** veya bir durum bildirmiyorsa **'UNKNOWN'** değerini alır.
- **UNFORMATTED=formatsız** , *formatsız* bir alfasayısal değişken olup kütüğün formatsız olup olmadığım bildirir. Formatsız ise **'YES'**, değilse **'NO'** veya bir durum bildirmiyorsa **'UNKNOWN'** değerini alır. Aşağıdaki seçime bağlı belirteçlerin bir değer alabilmesi için *mevcut* = .TRUE. ve *açıkmı* = .TRUE. olmalıdır.
- **NUMBER=sayı**, kütüğün birim numarasını verir.
- **ACCESS=erişim**, *erişim* 'SEQUENTIAL' (sıralı) veya 'DIRECT' (doğrudan) değerleri alabilen bir alfa sayısal değişkendir.

- **`FORM`=şekil**, *şekil* **`'FORMATTED'`** veya **`'UNFORMATTED'`** değerleri alabilen bir alfasayısal değişkendir. Aşağıdaki seçime bağlı belirteçlerin bir değer alabilmesi için *mevcut* = `.TRUE.` ve *erişim* = `DIRECT` olmalıdır.
- **`RECL`=kayıt**, kayıt tamsayı değişkeni olup kullanılan kayıdın uzunluğunu verir.
- **`NEXTREC`=sonraki-kayıt**, ulaşılan kayıttan bir sonraki kayıt numarasını verir. Eğer *mevcut*=`.TRUE.` ve *formatlı*=`FORMATTED` ise `BLANK`=boşluk belirteci kullanılabilir.
- **`ACTION`=action**, açılan kütüklerin **`'READ'`**, **`'WRITE'`**, **`'READWRITE'`** ve açılmamış kütükler için **`'UNDEFINED'`** değerlerini alır.
- **`READ`=oku**, kütüğün sadece okuma erişimli olarak açılıp açılamayacağını gösterir ve `'YES'`, `'NO'` veya `'UNKNOWN'` değerlerini alır.
- **`WRITE`=yaz**, kütüğün sadece yazma erişimli olarak açılıp açılamayacağını gösterir ve **`'YES'`**, **`'NO'`** veya **`'UNKNOWN'`** değerlerini alır.
- **`READWRITE`=okuyaz**, kütüğün hem okuma hem de yazma erişimli olarak açılıp açılamayacağını gösterir ve **`'YES'`**, **`'NO'`** veya **`'UNKNOWN'`** değerlerini alır.
- **`DELIM`=delim**, kütükte karakter belirteci kullanımını gösterir ve **`'APOSTROPHE'`** `(')`, **`'QUOTE'`** `(“)`, **`'NONE'`** veya **`'UNKNOWN'`** değerlerini alır.
- **`PAD`=pad**, kütüğün boşluklarla desteklenim desteklenmediğini açıklar ve **`'YES'`** veya **`'NO'`** değerlerini alır.
- **`IOLENGTH`=uzunluk**, formatsız bir kütüğün kayıt uzunluğunu gösterir.

Tablo 4.2 Kütük tiplerinin karakteristikleri

	Karakteristikler	
	Formatlı	*Formatsız*
Sıralı Erişimli	ASCII karakterler RETURN ile sona erer. Değişken-uzunluklu kayıtlar DOS cihazları ve disk kütükleriyle beraber kullanılabilir Ekranda kolaylıkla izlenebilir veya çıktısı alınabilir. Kayıtlar sırayla işlenir. Genellikle daha yavaş	İkilik veri Kayıt işareti ile ayrılır. Değişken uzunluklu kayıtlar Sadece disk kütükleriyle kullanılabilir. Kolaylıkla görüntülenemez ve işlenemez Kayıtlar sırayla işlenir Formatlıdan daha hızlı okuma ve yazma
Doğrudan Erişimli	ASCII karakterler Sabit kayıt uzunluğu 0 kayıt satır başıdır Kolaylıkla görüntülenemez ve işlenemez Kayıtlara herhangi bir sıra ile Formatlı sıralı disk kütüklerinden daha hızlı	ikilik veri Sabit kayıt uzunluğu 0 kayıt satır başıdır Kolaylıkla görüntülenemez ve işlenemez Kayıtlara herhangi bir sıra ile erişilebilir erişilebilir G/Ç da en hızlı

ÖRNEK 2: Çıktı kütüğü olarak açılması istenen bir dosyanın bilgisayarda mevcut olup olmadığını kontrol eden, mevcut ise üzerine yazma işleminin yapılıp yapılmayacağı hakkında kullanıcıyı sorgulayan bir program yazınız.

Bu programı yazarken **INQUIRE** deyimden yararlanmak gerekir.

```
PROGRAM Ornek2
IMPLICIT NONE
CHARACTER(LEN=20) :: kutuk_ismi
CHARACTER :: EH
LOGICAL   :: Mevcut_mu
LOGICAL   :: Acik_mi= .FALSE.
! Kütük açılıncaya kada DO döngüsünde çevir
KutukAc: DO

   ! Çıktı kütüğünün ismini oku
   WRITE (*,*) 'Çıktı kütüğü ismini giriniz: '
   READ (*,'(A)') Kutuk_ismi
   ! Kütük diskte mevcut mu?
   INQUIRE ( FILE=Kutuk_ismi, EXIST=Mevcut_mu )

   mevcut: IF ( .NOT. Mevcut_mu ) THEN
   ! Kütük diskte mevcut değil, yazma amacıyla açılabilir.
      OPEN(UNIT=9,FILE=Kutuk_ismi,STATUS='NEW', &
            ACTION='WRITE')
      Acik_mi = .TRUE.        ! Artık kütük açılmış durumda
    ELSE
   ! Kütük diskte mevcut.  Üzerine yazılsın mı?
      WRITE (*,*) 'Çıktı kütüğü mevcut. '
      WRITE (*,*) 'Üzerine yazalım mı? (E/H)'
      READ (*,'(A)') EH
     IF( EH=='h') EH='H'      ! Cevabı büyük harfe dönüştür
     IF( EH=='e') EH='E'
      yaz: IF ( EH == 'H' ) THEN
   ! Üzerine yazmak üzere kütüğü aç.
         OPEN(UNIT=9, FILE=kutuk_ismi, STATUS='REPLACE', &
               ACTION='WRITE')
         Acik_mi = .TRUE.  ! Artık kütük açılmış durumda
      END IF yaz
   END IF mevcut

   IF (Acik_mi) EXIT        ! Acik_mi=.TRUE. ise çık

END DO KutukAc
! Şimdi çıktılar kütüğe kaydedilebilir
! kütük kapatılıp veriler korumaya alınabilir.
WRITE (9,*) 'Bu bir çıktı kütüğüdür!'
CLOSE (9,STATUS='KEEP')
END PROGRAM Ornek2
```

4.6 NAMELIST I/O DEYİMİ

NAMELIST I/O deyiminin kullanımı sabit bir değişken isimlerini okuma ve yazmanın kolay bir yoludur. Daima aynı değişken gurubu içinde okunan veya yazılan listeyi içerir.

Genel kullanım şekli

```
NAMELIST / Gurup_ismi_1 / Değişken_1 [, Değişken_2, ...]
```

olarak verilmektedir. Burada *Gurup_ismi_1* isim listesinin adı olup bir programda birden fazla, ancak farklı isimlerde, gurup kullanılabilir. Programda, ilk icra edilebilir deyimden önce kullanılmalıdır. Değişkenler arzulanan sırayla (kütükteki sıraya karşı gelmek kaydıyla) art arda virgül ile ayrılarak dizilir.

İsim listesi kullanımı, formatlı girdi/çıktı kullanımını andırır. Sadece **FMT** yerine **NML** kullanılır.

Programda genel kullanım şekli

```
WRITE (UNIT=u,NML=Gurup_ismi_1, [...])
READ (UNIT=u,NML=Gurup_ismi_1, [...])
```

olarak verilir.

Aşağıdaki programda bir isim listesinin kütüğe yazdırılması ele alınmıştır. Bu program çıktısında isim listesinin adını (&LISTE) takiben her bir değişkenin ismi ve aldığı değerlerin virgül ile ayrılarak belirli bir alana sığdırarak formatsız bir şekilde verildiğine dikkat ediniz.

```
PROGRAM IsimListesi
IMPLICIT NONE
INTEGER :: i = 11, j = 32
REAL :: a = -999., pi = 3.14159
CHARACTER(LEN=12) :: karakter = 'Denemedir'
NAMELIST / liste / i, j, karakter, a, pi
OPEN (8,FILE='CIKTI.DAT',DELIM='APOSTROPHE')
WRITE (UNIT=8, NML=liste)
CLOSE (8)
END PROGRAM IsimListesi
```

Programının çıktısı aşağıda verilmiştir.

```
&LISTE I=11,J=32,KARAKTER='Denemedir',A=-999.000000,PI=3.14159012
```

NAMELIST deyimi ile veri okunması, yazdırma işlemi ile aynı şekilde yapılır. Aşağıdaki örneği inceleyiniz.

```
PROGRAM isim_listesi_oku
IMPLICIT NONE
INTEGER :: i = 11, j = 32
REAL :: a = -999., pi = 3.14159
CHARACTER(LEN=12) :: karakter = 'Karakter '
NAMELIST / liste / i, j, karakter, a, pi
OPEN (7,FILE='CIKTI.DAT',DELIM='APOSTROPHE')

! Güncellemeden önce NAMELIST yazdır
WRITE (*,'(1X,A)') 'Güncellemeden önce isim listesi :'
WRITE (UNIT=*, NML=liste)
READ (UNIT=7,NML=liste)

! Güncellemeden sonra NAMELIST yazdır
WRITE (*,'(1X,A)') 'Güncellemeden sonra isim listesi:'
WRITE (UNIT=*, NML=liste)

END PROGRAM isim_listesi_oku
```

Programının çıktısı aşağıda verilmiştir.

```
 Güncellemeden önce isim listesi :
  &LISTE I=11,J=32,KARAKTER='Denemedir',A=-999.000000,PI=3.14159012
 Güncellemeden önce isim listesi :
 &LISTE I=11,J=32,KARAKTER='Karakter ',A=-999.000000,PI=3.14159012
```

ÖRNEK 3: Aşağıdaki programda `A=1`, `B=2.`, `C=3.`, `X=4.` ve `Y=5.` olduğunda program çıktısını bulunuz.

```
PROGRAM Ornek3
IMPLICIT NONE
REAL :: A, B, C, X, Y
REAL, DIMENSION(4):: D=(/1.2, 3.4, 5.1, 7.6/)
NAMELIST/ABC/A, B, C,  /LIS/X, Y, D
READ(6,*) A, B, C, X, Y
WRITE(6,ABC)
WRITE(6,LIS)
END PROGRAM Ornek3
```

Program çıktısı

```
&ABC A=1.00000000,B=2.00000000,C=3.00000000/
&LIS X=4.00000000,Y=5.00000000,D=1.20000005,3.40000010,5.09999990,7.59999990/
```

şeklinde elde edilir.

4.7 HARİCİ KÜTÜKLER

Harici kütükler, disk kütükleri veya disk-olmayan kütükler şeklinde ikiye ayrılırlar. Bu kısma kadar bahsedilen kütük açma, kapama v.b işlemler disk kütükleri için verilmiştir. Bu kısımda özellikle bahsetmek istediğimiz konu disk-olmayan kütük kullanımıdır ki bunlar cihaz olarak adlandırılır. Cihaz ya konsol (monitör) veya işletim sistemi ile tanımlanmış bir cihaz kütük ismi olabilmektedir. Bunlar; CON (konsol), AUX (yedek), COM1 (iletişim 1), COM2 (iletişim 2), LPT1, LPT2, LPT3, PRN (yazıcı) veya NUL (boş) olabilmektedir ve FILE= ile kullanılırlar.

ÖRNEK 4: Doğrudan erişimli formatlı kütük kullanımını içeren bir programın hazırlanması.

```
PROGRAM Ornek4
IMPLICIT NONE
INTEGER :: i
INTEGER :: irec
CHARACTER(LEN=40) :: satir

! Bir kayıt başına 40 karakterden oluşan doğrudan erişimli bir kütük aç
OPEN(UNIT=11,FILE='dek.fmt',ACCESS='DIRECT',   &
     FORM='FORMATTED',STATUS='REPLACE',RECL=40)

! Bu kütüğe 100 adet kayıt girişi yapılsın
DO i=1,100
   WRITE(11,'(A,I3,A)',REC=i) i, ',ci kayıt girişi'
END DO

! Kullanıcı hangi kayıdı görmek istiyor?
WRITE(*,'(A)') 'Hangi kayıdı görmek istiyorsunuz?'
READ(*,'(I3)') irec

! Arzulanan kaydı getir
READ(11, '(A)',REC=irec) satir

! Kaydı ekranda görüntüle
WRITE(*,'(A,/,5x,A)') 'Kayıt',satir

END PROGRAM Ornek4
```



Sıralı erişimli kütüklere veriler sırayla yazılır ve bu sırayla okunur. Doğrudan erişimli kütüklerde, kütükteki herhangi bir veri kaydına ulaşılabilir veya değiştirilebilir.

ALIŞTIRMALAR

4.1 Aşağıdaki Fortran deyimler gurubunda hata varsa hataları belirleyiniz.

```
(a)   INTEGER :: n=15
      OPEN(UNIT=n,FILE='Dosya.Dat',STATUS='SCRATCH')
      WRITE(UNIT=n,*) d1,d2

(b)   OPEN(3,'Dosya.dat','Old')

(c)   OPEN(Unit=24)

(d)   OPEN(UNUT=1.2E4,FILE='s.dat')

(e)   OPEN(Unit=25,file=”abc.dat”,STATUS='UNKNOWN')

(f)   OPEN(UNIT=-11,STATUS='OLD')

(g)   OPEN(999,FILE='SCRATCH',STATUS='NEW')
```

4.2 Bir mağazada 435 çeşit ürün satılmaktadır. Mağazaya mal giriş-çıkışının bilgisayarlı bir sistem ile yapılması arzu edilmektedir. Ürün adı, fiyat ve stok verileri bir kütükten girilmektedir. Stoğu tükenen ürünün varlığını araştıran bir program yazınız.

4.3 Periyodik tablodaki elementlerin ismi, kısaltılmış ismi, atom numarasından oluşan sıralı erişimli bir kütük (veri tabanı) oluşturan program yazınız. Bir elemente ait özellikleri `NAMELIST` ile temsil ediniz.

4.4 Aşağıda verilen koşulları yerine getirecek Fortran deyimlerini yazınız. Bir kayıda bir gerçek sayının yazıldığı `GIRDI.DAT` kütüğü mevcut olduğunu kabul ediniz.
a) Birim numarası 15 olan `GIRDI.DAT` isimli mevcut kütük ile birim numarası 50 olan `CIKTI.DAT` isimli kütüğü okuma ve yazmaya açınız.
b) `GIRDI.DAT`'dan kütük sonuna ulaşıncaya kadar verileri okuyunuz. Verilerden pozitif olanları `CIKTI.DAT` kütüğüne kaydediniz.
c) Girdi ve çıktı kütüklerini kapatınız.

4.5 Bir dosyadan 500 adet gerçek sayı okuyan ve aritmetik ortalamasını hesaplayan bir program yazınız.

4.6 Bir dizi gerçek sayıyı kullanıcı-tarafından hazırlanmış bir kütükten okuyan, bu sayısal değerleri noktadan sonra iki basamak olarak çıktı kütüğüne kaydeden bir program yazınız.

4.7 Kütük ismi `Veri.dat` olan kütükten art arda dizili sonlu sayıda tamsayı bulunmaktadır. Bu sayıları kütük sonuna kadar okuyan, sayıların adedini, pozitif olanların adedini hesaplayan bir program yazınız.

4.8 Bir kütükten bir dizi gerçek sayı okunmaktadır. Kütük sonuna ulaşıldığında 5 satır geriye gelerek verileri tekrar okuyan bir program parçası hazırlayınız.

4.9 Aşağıdaki deyimleri inceleyiniz ve varsa hatalarını tespit ediniz.

```
a) OPEN(11,FILE='A.DAT',STATUS='new',IOSTAT=hata)
   READ(11,*) x, y, z, xy1

b) OPEN(UNIT=70,FILE='G.DAT',STATUS='STRACTH',IOSTAT=hata)

c) OPEN(UNIT=88,FULE='VERI',STATUS='NEW',&
         ACTION='READWRITE',IOSTAT=hata)

d) INTEGER :: birim=25
   OPEN(UNIT=birim,FILE='A.KUT',STATUS='OLD',IOSTAT=hata)
   READ(25,*) birim
   CLOSE(UNIT=birim)

e) OPEN(UNIT=45,FILE='C.TXT',STATUS='NEW', &
          ACTION='WRITE', IOSTAT=hata)
   WRITE(45,*) A, B, C
   WRITE(45,*) X, Y
   CLOSE(45)
```

4.10 Aşağıdaki programların çıktılarını bulunuz

(a)

```
PROGRAM Soru_a
IMPLICIT NONE             ! AA kütüğündeki veriler
REAL :: A, B, C, D, X     ! 7.  0.  4.  5.
REAL :: x1, x2, x3, Y     ! 9.  3.  2.
REAL :: Y1, Y2
OPEN(3,FILE='AA')
READ(3,*) A,B,C,D
X=3.*A+B-C+4.*D
REWIND 3
READ(3,*) X1,X2,X3
Y=(Xl+X2+X3)/3.
REWIND 3
READ(3,*) Y1,Y2
Y1=Y1+A
PRINT*, X,Y,Y1,Y2
END PROGRAM Soru_a
```

(b)

```
PROGRAM Soru_b             ! Gir kütüğündeki veriler
REAL :: A, B, C, D         ! 3.  5.
OPEN(2,FILE='GIR')         ! 4.  6.
I=0                        ! 4.  4.
12 READ(2,*) A, B          ! 6.  4.
I=I+1                      ! 5.  3.
IF(B<A) THEN               ! 5.  4.
     D=B                   ! 3.  5.
     C=C+D                 ! 1.  6.
ELSE                       ! 4.  7.
     D=A                   ! 8.  2.
ENDIF
WRITE(*,4) A, B, D
IF(I/=10) GO TO 12
WRITE(*,4) C
4 FORMAT(3(2X,F5.2))
END PROGRAM Soru_b
```

(c)

```
PROGRAM Soru_c
REAL :: X=3., Y=5., Z=11., P=45.67
REAL :: DX, A, B, U
INTEGER :: JKL
OPEN(88,FILE='ABC.DAT',STATUS='NEW')
A =X
DX=0.1*B
DO WHILE (JKL<20)
   A=A+DX
   JKL=JKL+1
IF(A<5.) THEN
   U=Z+0.2*P+0.33*Y
ELSE
   U=l.2*Z-0.04*P+0.33*Y
ENDIF
IF(ABS(U)>5.) U=U-6.0
DX=DX+U
WRITE(88,30) A,U,DX
END WHILE
30 FORMAT(/,5X,'=',F10.6,5X,'U='/,T40,'X=',F12.8/)
END PROGRAM Soru_c
```

BÖLÜM 5
FORMATLI GİRDİ/ÇIKTI HAZIRLAMA

Bir program yazıldığında, algoritmanın programlanması kadar zaman alan bir bölüm de verilerin ekrana veya kütüğe aktarılmasında yapılacak düzenlemelerdir.

Bir çıktı düzeni hazırlanırken, bu çıktıdaki veriler ile ilgili şu soruların yanıtlanması gerekir: Veriler tamsayı, gerçek sayı, alfa sayısal mıdır? Üstel olmayan yada üstel gerçek sayı mıdır? Gerçek sayıların ondalık noktasından sonra kaç basamağı yazdırılmak isteniyor? Sayfa düzeni nasıl olsun? Alfa sayısalların alan uzunluğu kaç olmalıdır? v.s. İşte bu gibi kararların bir çoğu `FORMAT` deyiminin kullanımı ile alınır. Programlamanın, belki de, en sıkıcı tarafı girdi ve çıktı değerlerinin düzenlemesi, yani formatlanması, kısmıdır.

5.1 FORMATLI G/Ç DEYİMLERİ

Bu noktaya kadar bahsedilen G/Ç (**I/O**) deyimleri ekrana hitap eden, asteriks (`*`) kullanımını içeren, `READ` ve `PRINT` deyimleriydi. Bu deyimlerin ekranla girdi/çıktı alışverişi dışındaki kullanım alanlarına bu kısımda verilmektedir.

Bir formatlı girdi/çıktı deyiminin girdi veya çıktı olarak tanımlanabilmesi için dört koşulun yerine gelmesi gerekir:

1. **`READ`** veya **`WRITE`** deyimlerinden birinin kullanımı, yani

   `READ` `(Cihaz,Format)` *Değişken Listesi*

   veya

   `WRITE``(Cihaz,Format)` *Değişken Listesi*

   şeklindedir.
2. Kullanılacak cihaz (disk, optik okuyucu, yazıcı, v.s) seçimi; FORTRAN da cihazlara l, 2, 3,..., n gibi *birim* (ünite) *numarası* atanarak yapılır; sistem bilgisayarlarında (main frame bilgisayarlarda) girdi ve çıktı birimi numaraları sırasıyla 5 ve 6'dır.
3. Girilecek veya çıktısı alınacak verilerin düzenini tanımlayan **`FORMAT`** deyiminin hazırlanması ve kullanılması,
4. Değişkenlerin girişi/çıkışının, **`FORMAT`** sırasına uygun bir şekilde kullanılması (listelenmesi).

Aşağıda FORTRAN'da kullanılan `READ` deyiminin genel kullanım şekli verilmektedir:

```
READ(kontrol_Listesi) Girdi Değişken listesi
```

Burada *`kontrol_Listesi`* virgül ile ayrılmış birden fazla belirteçten oluşur. `READ` deyimi ile kullanılan belirteçler ve işlevleri Tablo 5.1'de verilmektedir:

Tablo 5.1: READ/WRITE deyimlerinde kullanılan belirteçler.

Belirteç	Kullanım	Amaç	Muhtemel Değerler
[**UNIT**=]*tamsayı değişkeni*	*Girdi*	Girdi/Çıktı birimi numarası	İşlemciye bağlı tamsayı
[**FMT**=]*deyim numarası* [**FMT**=]*karakter ifadesi* [**FMT**=]*****	*Girdi*	Formatlı veri okunurken, verilerin yazdırılma düzenini belirlemeye yarar	
IOSTAT=*tamsayı değişkeni*	*Çıktı*	Okuma işlemi sonunda I/O statüsünü belirtir.	Başarılı = 0 Başarısız= pozitif tamsayı Kütük sonu= -1 Kayıt sonu= -2
REC=*tamsayı değişkeni*	*Girdi*	Doğrudan erişimli kütükte okunacak kayıt numarasıdır	
NML=*İsim listesi adı*	*Girdi*	Okunacak isim listesini belirtir	
ADVANCE= *karakter ifadesi*	*Girdi*	İlerleme veya ilerleme olmayan I/O işlemini belirtir	Sıralı erişimli kütüklere hastır 'YES','NO' değerleri alır
SIZE= *tamsayı değişkeni*	*Çıktı*	İlerleme olmayan I/O işleminde okunacak karakter sayısını belirtir.	

READ deyiminin alternatif bir kullanım şekli de aşağıda verilmektedir.

```
READ FMT=format_no, <Girdi listesi>
```

Bu deyim standart girdi cihazı (kişisel bilgisayarlarda bu cihaz genelde klavyedir; **UNIT=*** veya **UNIT=5**) için kullanılır. **FMT=*format***, verilerin okunduğu format etiketini belirtir. Format etiketi yerine, daha kısa bir format ise, formatın kendisi tırnak işareti arasında yazılır. Bazı örnekler,

READ*, <girdi listesi>	Değişkenler virgül ile ayrılır; girdi ekrandan okunur.
READ(10,*)<girdi listesi>	Yukarıdaki ile aynı, yalnız girdi 10 birim numaralı kütükten okunur.
READ(5,*)<girdi listesi>	OPEN deyimi ile birim numarası 5 olan girdi kütüğü açılmamışsa, READ*, deyimi ile aynı görevi görür.
READ(*,*)<girdi listesi>	Deyim READ*, ile aynı görevi görür.
READ 8, <girdi listesi>	Girdi değerleri ekrandan 8 etiket numaralı formata göre okunur.
READ(2,4)<girdi listesi>	Girdi listesini birim numarası 2 olan kütükten etiket numarası 4 olan format düzeninde okur.

```
READ FMT=10, x, y
READ 10, x, y
10 FORMAT(2F9.3)
```

Etiket numarası 10 olan format düzeninde x ve y'nin değerlerini okur.

READ '(2F9.3)', x,y — Yukarıdaki deyimin tek satırda verilmesidir.

Alfa='(2F9.3)'
READ Alfa, x,y — Yukarıdaki örnekle aynı işleve sahiptir.

READ(**11**,'(3i5)',**REC**=kayit) deger — Sıralı erişimli kütükten kayit kaydında bulunan bilgileri (deger), (3i5) formatında okur.

Çıktı düzenleme amacıyla kullanılan **WRITE** deyiminin genel kullanım şekli

WRITE(*kontrol_Listesi*) *<Çıktı listesi>*

olarak verilmektedir. Aynı şekilde *kontrol_Listesi* virgül ile ayrılmış birden fazla belirteçten oluşur. Bu belirteçler READ deyimi için verilenler ile aynıdır.

Çıktı amacıyla kullanılan alternatif bir deyimde PRINT olup, genel kullanım şekli de aşağıda verilmiştir:.

PRINT FMT=*format_no*, *<Girdi listesi>*

Bazı örnekler,

PRINT*, <çıktı listesi> — Listeli değişkenler formatsız olarak ekrana yazılır.
WRITE(8,*) <çıktı listesi> — Listeli değişkenler 8 birim numaralı çıktı kütüğüne yazılır.
WRITE(*,*) <çıktı listesi> — PRINT*, deyimi ile aynı işleve sahiptir.
WRITE(6,*)<çıktı listesi> — OPEN deyimi ile birim numarası 6 olan çıktı kütüğü açılmamışsa, PRINT*, deyimi ile aynı görevi görür.
WRITE(2,4)<çıktı listesi> — Çıktı listesini birim numarası 2 olan kütüğe etiket numarası 4 olan format düzeninde yazar.
WRITE(UNIT=2,FMT=4) <çıktı listesi> — Yukarıdaki örneğin alternatifidir.
WRITE(2,4,IOSTAT=durum) <çıktı listesi> — Yukarıdaki örneğin bir alternatifi olup yazdırma esnasında bir hata oluşup oluşmadığı IOSTAT ile kontrol edilir. durum=0 başarılı yazdırma olduğunu belirtir

PRINT 10, a, b
10 FORMAT(2F9.3) — Format numarası 10 olan format düzeninde x ve y'nin değerlerini yazar.
PRINT '(2F9.3)', a, b — Yukarıdaki deyimin tek satırda verilmesidir.

Alfa='(2F9.3)'
PRINT Alfa, a,b — Yukarıdaki örnekle aynı işleve sahiptir.

WRITE(**11**,'(3i5)',**REC**=kayit) deger — Sıralı erişimli kütüğen kayit kaydı olarak bilgileri (deger), (3i5) formatında yazar.

Yukarıda kullanılan girdi/çıktı deyimlerinde, asteriksler yerine formatın kendisi, format numarası veya etiketi yazılarak, girdi veya çıktı düzeni formatlı olarak belirlenir.

5.2 FORMAT DEYİMİ

FORMAT deyiminin genel kullanım şekli aşağıda verilmektedir:

```
etiket FORMAT(belirteç1, belirteç2, belirteç3,. . .)
```

Burada *etiket* bir tamsayı olup, bu sayıya *format numarası* (veya *format etiketi*) denir; FORMAT kelimesi programda herhangi bir sütundan itibaren başlayabilir. Parantez içinde yer alan terimler, format belirteçlerinin bir listesidir. Bu belirteçler virgül ile (veya "/" işareti ile) birbirinden ayrılır.

FORMAT deyimi çalıştırılabilir bir programlama deyimi olmadığından (*nonexecutable*) programın herhangi bir yerinde (READ/WRITE deyimlerinin kullanıldıkları yerlerden bağımsız olarak) kullanılabilirler. FORMAT deyimiyle üç unsur tanımlanır:

1. **Belirleyici Tipi.** Belirteçler değişken veya sabitin gerçek, tamsayı, alfa sayısal v.s tipine göre değişir; READ/WRITE deyimlerindeki değişken listesi tipi ile uyumlu belirteçler kullanılır.

2. **Alan uzunluğu.** Sayısal değerin karakter olarak işgal ettiği veya işgal etmesi arzulanan azami uzunluktur.

3. **Ondalık noktasının yeri.** Ondalık noktasının yeri alanın sağından itibaren kaç karakter geriye yerleştirileceği belirtilir (sadece REAL tipindeki sayılar için kullanılır).

Format belirteçleri genelde dört guruba ayrılabilir

1. Dikey konumlamanın yapıldığı format belirteçleri,
2. Yatay konumlamanın yapıldığı format belirteçleri,
3. Bir değerin çıktıdaki formatını düzenleyen belirteçler,
4. Bir formatı veya format parçasını tekrarlı bir şekilde düzenleyen belirteçler

5.2.1 FORMAT BELİRLEYİCİLERİ-*SAYISAL BELİRTEÇLER*

5.2.1.1 GERÇEK SAYILAR—F FORMAT BELİRTECİ

"F" format belirtecinin kullanım şekli **Fw.d** veya **fw.d** olarak verilmektedir; burada **w** alan uzunluğudur (yani sayının içine sığdırılacağı toplam karakter sayısıdır); bu uzunluğa ± işareti ve ondalık noktası da dahil edilir, **d** ise ondalık noktasından sonra arzu edilen hassas basamak sayısını ifade etmektedir. Örneğin, A=–1245.127 sayısının çıktısını eksiksiz bir şekilde ve formatlı olarak almak istersek, bu sayının karakter olarak uzunluğunu 9 (eksi işareti için l, 1245 için 4, ondalık noktası için 1 ve 127 için 3 karakter) ve ondalık noktasından sonraki hassas basamak sayısını 3 buluruz. Bu durumda, format belirleyicisi kendiliğinden oluşur: yani **F9.3** olmalıdır.

Bir başka örnek daha ele alalım: bir programdan parçası aşağıdaki gibi verilmiş olsun.

```
REAL :: x1, Y, Z
xl=-2.483501
Y=161.0052
Z=4.0
WRITE(*,9) x1,Y,Z
9 FORMAT(F11.6,F12.4,F5.2)
```

Bu programın çıktısı ekranda aşağıdaki gibi gözükür:

```
         1         2         3
123456789012345678901234567890
  -2.483501    161.0052 4.00
```

NOT: *Yukarıdaki çıktıda verilen numaralar bilgisayar çıktısında gözükmez; sadece çıktıdaki sütun numarasını belirtmek için bir şablon olarak değerlendirilmelidir.*

Formatlı Girdi/Çıktı da dikkat edilecek bazı hususlar şöyle sıralanabilir:

1. Hem ekrana hem de yazıcıya gönderilen çıktılarda, her satırın ilk sütunu gözükmeyebilir. Bilgisayar sistemlerinin çoğu bir çıktı satırının ilk sütununu çıktının dikey olarak konumlandırılmasında (*Carriage Control*) kullanır. Kısacası formatın birinci sütuna çıktı gelmeyecek şekilde hazırlanmalıdır!
2. **REAL** sayılar ile çıktı alınırken, çıktıdaki değerler sayıyı belirli bir basamaktan ötesini kesme işleminden ziyade yuvarlama yaparlar. Örneğin, 1.867 sayısının **F6.2** ile çıktısı, ondalık noktasından sonra iki basamağı göz önüne alacaktır. Bu durumda, çıktı 1.86 değil 1.87 olarak elde edilir; yani 8'den sonraki 67 kısmı 70'e yuvarlanır.
3. Normalde, veriler okunurken bilgisayar sayılar arasındaki boşlukları boşluk olarak okur. Buna rağmen bazı bilgisayar sistemleri okuma işleminde boşlukları sıfır olarak algılar. Bu durumlar ile karşılaşmamak için programa girilen değerlerin belirtilen formata uygun olması gerekir.
4. **READ** ile okunan değer, alan uzunluğuna sığmıyor ve ondalık noktası içeriyorsa, bu durumda ondalık noktasının yeri "**d**" ile belirlenen uzunluğu aşar. Örneğin, 974.735 sayısı **F10.3** olarak verilmiş olsun; bu durumda girdi:

```
         1         2
12345678901234567890
 974.73500
```

olarak, yani **F10.5** şeklinde, görülür.

5. Okunan bir sayısal değer ondalık noktası içermiyorsa, F formatındaki **d** belirteci 10'nun negatif kuvveti şeklinde uygulanacaktır. Örneğin, 1234 sayısı **F5.2** formatı ile okunacak olsun; bu durumda okunan değer

$$1234.\times10^{-2} \qquad (d=2 \text{ için})$$

olacaktır. Bu nedenle, gerçek sayılara daima ondalık noktasının ilave edilmesi unutulmamalıdır.

6. Sayısal değerler, belirteci ile belirtilen alanının sağından itibaren hizalanır. Ancak gerçek sayıların girdi olarak kullanıldığında ondalık noktasının yeri, format belirleyicisiyle belirtilen yerde olsun veya olmasın, sayısal değerin doğru okunmasına neden olur.

5.2.1.2 TAMSAYILAR—I FORMAT BELİRTECİ

I format belirtecinin genel kullanım şekli "**Iw**" veya "**iw**"dir; burada **w** alanın toplam uzunluğudur. Bu belirteç, tamsayılar için kullanıldığından, ondalık noktasının yerini belirtmeye gerek yoktur. Örneğin,

```
INTEGER :: n=27
READ(5,12) n
12 FORMAT(I9)
```

program parçasına girdi değeri aşağıdaki şekilde sağlanmalıdır; yani satırın solundan itibaren 9 karakterlik alan ayrılır, 27 sayısı en sağ taraftan hizalanarak yazılır. Bu işlem 27'nin 8 ve 9.cu sütunları işgal edecek şekilde yazılmasını gerektirir.

```
         1
1234567890
       27
```

Eğer bu sayı aşağıdaki gibi yanlış format ile girilirse (4 ve 5.ci sütunlara yazılsa),

```
         1
1234567890
   27
```

boşlukları “boşluk” olarak algılayan bilgisayarlarda, sayı doğru okunur; fakat boşlukları "sıfır" olarak yorumlayacağından bu sayı 270000 olarak algılanır. (**DİKKAT:** *Bu problem* OPEN *deyiminde* BLANK *kullanımı ile çözümlenebilir*. BLANK='NULL' *boşlukları “boşluk” olarak algılar;* BLANK='ZERO' *ise boşlukları “sıfır” olarak algılar. Bu nedenle* OPEN *deyimi ile beraber* BLANK *belirtcinin kullanımına dikkat edilmelidir*.)

Sayıların bir kütükten programa formatlı olarak teminine ilişkin bazı örnekler aşağıda verilmektedir (sol sütunda kütükteki sütun sırasını vermektedir).

```
         1          Format         READ işlemi
1234567890          Belirteci      Sonucu
-------------------------------------------------
      5432          I4             0000
      5432          I10            5432
5432 123            I4, I4         5432 ve 0123
  1234              I10            12340000
    123.45          F5.2, F5.2     0.01 ve 23.45
    123.45          F10.2          123.45
 123.45             F10.0          0123.4500
-------------------------------------------------
```

REAL ve INTEGER sayıların yazılması (WRITE ile) sırasında ortaya çıkan sayıların içine sığabileceği alan uzunluğu yeterli olmalıdır. Eğer alan genişliği **w** yetersiz ise alan uzunluğu asteriks (*) ile doldurulur. (*Ancak farklı derleyiciler başka karakter kullanabilir!)* Örneğin, A=678.901 ve I=12345 sayılarının, sırasıyla F5.3 ve I4 formatına göre, çıktıları ***** ve **** ile sonuçlanır.

Aşağıda tamsayıların formatlı çıktılarına ilişkin bazı örnekler verilmiştir:

```
                    Format          WRITE sonucu
                    Belirteci                 1
    Tamsayı                          1234567890
-------------------------------------------------
-123                I6                  -123
88                  I10                     88
 786                I10                    786
-23456              I5                   *****
-------------------------------------------------
```

Burada `-23456` sayısı, **`I5`** alanının içine sığamadığından çıktısında `*****` yazılmasına neden olmuştur.

Görüldüğü üzere programa formatlı girdi temininde format belirteçleri ile ortaya konan kurallara kesinlikle uymamız gerekir; aksi takdirde, programa veri okutma esnasında, bilgisayara yanlış veri temin etmiş oluruz. Bu durumu fark edinceye kadar oldukça fazla zaman kaybetmemek ve bir otomatik kontrol sağlamak amacıyla programa ekrandan veya kütükten girilen bütün verileri ekrana veya bir kütüğe yazdırınız; girdilerin doğruluğunu kontrol ediniz. Verilerin programa sağlıklı bir şekilde temin edildiğinden emin olduktan sonra, programdaki ekrana yazdırma deyimlerini iptal edebilir veya silebilirsiniz.



Programa ekrandan veya kütükten girilen tüm verileri, ekrana veya çıktı kütüğüne yazdırınız. Girilen veriler ile çıktıdakilerin aynı olup olmadığını kontrol ediniz.

5.2.1.3 GERÇEK SAYILAR—E FORMAT BELİRTECİ

Bilimsel (*scientific*) sayı formatı olarak bilinen "**E**" format belirleyicisinin kullanım şekli **`Ew.d`** olarak verilmektedir; burada **`w`** ondalık noktası, sayının işareti (`+` veya `-`) ve üstel kısım dahil olmak üzere sayının işgal ettiği karakter sayısıdır; **`d`** ise ondalık noktadan sonra arzu edilen basamak sayısıdır. Üstel şekilde $a \times 10^n$ veya eksponesiyelli bir gerçek sayı, örneğin -0.12345×10^{-6}, tipik olarak `-0.12345E-06` şeklinde yazılır. Bu sayının alan uzunluğu (**`w`**) 12 ve noktadan sonraki basamak sayısı (**`d`**) 5 dir. Herhangi bir sayının çıktısında kullanılması `E` formatı için **`w`** $\geq$ **`d+7`** olmalıdır. Girdilerde kullanılan `E` formatının değeri alanın sağından itibaren hizalanarak yerleştirilir. Üstel kısımdaki rakam en fazla iki basamaklı tamsayı olmalıdır. Eğer `E` format belirteci unutulursa, bilgisayar bunu `F` formatı olarak algılar.

5.2.1.4 GERÇEK SAYILAR—ES FORMAT BELİRTECİ

FORTRAN 90/95'in yeniliklerinden olan "**ES**" format belirtecinin kullanım şekli **`ESw.d`** olarak verilmektedir; burada **`w`** ondalık noktası, sayının işareti (`+` veya `-`) ve üstel kısım dahil olmak üzere sayının işgal ettiği karakter sayısıdır; **`d`** ise ondalık noktadan sonra arzu edilen basamak sayısıdır. Kullanım şekli aynı **`E`** format belirtecinde olduğu gibidir; ancak aralarındaki fark **`E`** formatında 0 ile 1 arasında değişen bir sayı 10'un kuvveti ile çarpılı verilirken, **`ES`** formatında 1 ile 10 arasında değişen sayı çarpı 10'un kuvveti şeklinde verilir. Örneğin -0.12345×10^{-6} sayısı, **`E12.5`** format belirteci ile `-0.12345E-06` şeklinde yazılırken,

ES12.5 format belirteciyle -1.23450E-07 şeklinde yazılır. Aynı E format belirtecinde olduğu gibi, ES formatı için $w \geq d+7$ olmalıdır.

5.2.1.5 GERÇEK SAYILAR—EN FORMAT BELİRTECİ

FORTRAN 90/95'in yeniliklerinden olan *mühendislik* (*ENgineering*) "**EN**" format belirtecinin kullanım şekli **ENw.d** olarak verilmektedir; burada **w** ondalık noktası, sayının işareti (+ veya -) ve üstel kısım dahil olmak üzere sayının işgal ettiği karakter sayısıdır; **d** ise ondalık noktadan sonra arzu edilen basamak sayısıdır. Kullanım şekli aynı **ES** format belirtecinde olduğu gibidir; ancak aralarındaki fark **EN** formatında 1 ile 1000 arasında değişen bir sayı 10'un kuvveti ile çarpılı verilirken, **ES** formatında 1 ile 10 arasında değişen sayı çarpı 10'un kuvveti şeklinde verilir. Bu format belirtecinin mühendislikte kullanım nedeni 10^{-6}, 10^{-3}, 10^{3}, 10^{6} çarpımlarının birim dönüşümleri anlamları vardır. Örneğin, 1 gram=10^{-3} kg veya 1 kg=10^{3} gram gibi bir çok birimin bir biri cinsinden ifadesinde 1000'in katları söz konusudur.



Çok büyük veya çok küçük sayıların çıktısını alırken ES *formatını kullandığınızda sonuçlar daha anlaşılır olur.*

Aşağıdaki program parçası örneğini ele alırsak,

```
REAL :: a=0.00123, b=234.45, c=0.12345
WRITE(*,100)  a, b, c
WRITE(*,200)  a, b, c
WRITE(*,300)  a, b, c
100 FORMAT(E12.3, E16.5, E16.6)
200 FORMAT(ES12.3,ES16.5,ES16.6)
300 FORMAT(EN12.3,EN16.5,EN16.6)
```

programının çıktısı

```
         1         2         3         4
1234567890123456789012345678901234567890123456789012345
  0.123E-02     0.23445E+03    0.123450E+00
  1.230E-03     2.34450E+02    1.234500E-01
  1.230E-03   234.45000E+00  123.450000E-03
```

olarak elde edilir.

5.2.1.6 MANTIKSAL SAYILAR—L FORMAT BELİRTECİ

L format belirteci mantıksal verilerin çıktısında kullanılır ve **Lw** şeklinde verilir. Burada **w** mantıksal değişkenin görüntülenmesi için ayrılan alan uzunluğudur. Bir mantıksal değişkenin alabileceği değerler .TRUE. veya .FALSE.'dır. Mantıksal değişkenlerin çıktıları, w ile ayrılan alana sağdan hizalanarak ya T (*true*) yada F (*false*) olarak yazılır. Örneğin,

```
LOGICAL :: cikti=.TRUE., kapali=.FALSE.
WRITE(*,40) cikti, kapali
40 FORMAT(' ',L5,L7)
```

programının çıktısı

```
         1         2
12345678901234567890
    T      F
```

olarak yazılır.

5.2.1.7 GERÇEK SAYILAR—P Format Belirteci

Gerçek sayılı değişkenlerin format belirteçlerini etkileyen bir ölçü faktörü oluşturur. Kaç basamaktan sonra virgül ayrılacağını belirler. Genel kullanımı **nP** şeklinde olup **n** bir işaret (+ veya -) ve bir tamsayıdan oluşur. E, D, F, ES ve EN format belirteçlerinin *önüne* yazılarak kullanılır.

Girdi/çıktı deyimleri değerlendirilirken **n** ölçü faktörü sıfırdır. Ölçü faktörü bir kez belirlendi mi, program boyunca E, F ve ES format belirteçleri bu ölçü faktörüne uyarlar. Örneğin

```
100 FORMAT(F3.1,2PE11.4,I10,e18.14,f11.3)
110 FORMAT(-3PF8.2,F10.1,1PE12.5,e18.14,-1PF11.3,F12.2)
```

Formatlarından etiket numarası 100 olan formatta P'nin kullanıldığı ilk yer 2PE11.4 dür. Bunu takip eden diğer format belirteçlerden (F3.1 ise kullanımdan önce yer aldığı için) bundan etkilenmez iken, diğer belirteçler; yani E18.14 ve f11.3, **2P** belirtecinden etkilenir. Etiket numarası 110 olan formatta F10.1, **-3P**; E18.14, **1P** ve F12.2 de **-1P** belirteçlerinden etkilenirler.

n ölçü faktörü, çıktı değerlerinin 10^n ile çarpılmasına ve üstün n kadar eksiltilmesine sebep olur. Örnek,

```
REAL :: A=0.0012345, B=0.654321
WRITE(*,8) A,B
8 FORMAT(+2P,E14.5,2X,E14.5)
```

programının çıktısı

```
         1         2         3
123456789012345678901234567890
  12.3450E-04    65.4321E-02
```

olarak elde edilir. Bu çıktıda ondalık noktası ile E arasında 4 basamak olduğuna dikkat ediniz. Bunun nedeni ondalık noktasının solunda iki basamak olmasından kaynaklanmaktadır. Ondalık noktasının solunda üç basamak olsaydı; yani 8 no'lu formatta +2 yerine +3 koysaydık, bu durumda sayılar A=123.450E-05 ve B=654.321E-03 olurdu. Bu değerlerin A=1.23450E-03 ve B=6.54321E-01 olarak yazdırılması istenirse, bu durumda 8 numaralı format deyimindeki **+2P** belirteci **+1P** olarak değiştirilmelidir. Bazen bir format deyimi içinde P belirteci farklı birkaç durumda kullanılmak istenebilir; yani değerleri tek bir format deyimi ile A=123.450E-05 ve B=0.00654E+02 olarak yazdırmak istersek, bu durumda format deyimi aşağıdaki şekilde yazılabilir.

```
8 FORMAT(3PE14.5,2X,-2PE14.5)
```

Bu format tarzı ile birinci format belirleyicisi ondalık noktasının ilk üç rakamdan sonra konulmasını, ikincisinde de iki basamak geriye gidilerek konulmasını belirtmektedir.

5.2.8 DİĞER FORMAT BELİRTEÇLERİ

`B`, `O` ve `Z` format belirteçleri binray (ikilik), Octal (sekizlik) ve Hekzadesimal (on altılık) düzendeki sayıların formatında kullanılır. Kullanım şekilleri

```
Bw.d      Bw
Ow.d      Ow
Zw.d      Zw
```

olabilmektedir; burada `w` ve `d`'nin anlamları önceki format belirteçleri ile aynıdır.

```
INTEGER :: a=16, b=-1
WRITE(*,100) 'İkilik düzen   : ',a,b
WRITE(*,200) 'Sekizlik düzen : ',a,b
WRITE(*,300) 'Onaltılık düzen: ',a,b
100 FORMAT(1x,A,B16,1x,B16)
200 FORMAT(1x,A,O11.4,1x,O11.4)
300 FORMAT(1x,A,Z8,1x,Z8)
```

programının çıktısı

```
         1         2         3         4         5
12345678901234567890123456789012345678901234567890
İkilik düzen   :            10000 ****************
Sekizlik düzen :        0020 37777777777
Onaltılık düzen:       10 FFFFFFFF
```

olarak elde edilir. `B16` format belirteci `-1`'in temsil edilebilmesi için küçüktür; bu nedenle yazdırma alanı asteriks ile doldurulmaktadır.

5.2.2 KONUM TANIMLAYICILAR

5.2.2.1 DİKEY KONUMLAMA

FORTRAN'da yazıcıdan *kağıda* çıktı aktarırken, dikey konumlandırma yapmak mümkündür. Bunlar format deyiminin ilk sütununa yapılan karakter atamaları ile sağlanır.

`' '`	Boşluk	Kağıdı tek satır atlatarak çıktı verir
`'0'`	0 (sıfır)	Kağıdı çift satır atlatarak çıktı verir
`'1'`	1	Bir sonraki kağıdın başından itibaren çıktı verir
`'+'`	+	Kağıdın ileriye sarılmasını durdurur (Kağıt, çıktılar yazıldıktan sonra sabitlenir)

Bu dikey konumlayıcılar `'1'` gibi alfa sayısal olarak verilebilirler. Örneğin, aşağıda verilen 25 numaralı formatta `'-'` alfa sayısal sabit şeklinde tanımlanmıştır. İlk sütuna 0 (sıfır) veya 1 (bir) tamsayısının denk gelmesi de dikey konumlama olarak algılanır.

```
25 FORMAT('-',F9.4, v.s başka belirteçler )
```

ÖRNEK 1: Aşağıdaki programın kağıt üzerindeki çıktısını bulunuz.

```
PROGRAM dene
INTEGER :: n
OPEN(8,FILE='a.dat')
n=0
DO WHILE (n<14)
  n=n+2
  WRITE(*,100) n
END DO
100 FORMAT(i2)
END PROGRAM dene
```

Çıktı aşağıdaki şekilde gözükür:

```
12345                                   1. Kağıdın üst çizgisi
--------------------------------------------------------------
 2
 4
 6
 8
________________________________________2. Kağıdın üst çizgisi
 0
________________________________________3. Kağıdın üst çizgisi
 2
________________________________________4. Kağıdın üst çizgisi
 4
```

I2 formatı ile n'nin aldığı değerler (2, 4, 6, 8) ikinci sütundan itibaren hizalanarak yazılmaktadır. Birinci sütuna herhangi bir sayısal değer gelmediğinden bilgisayar bunu boşluk olarak yorumlamakta ve her değeri birer satır atlayarak yazmaktadır. n'nin 10, 12 ve 14 için değerlerini yazdırırken, çıktının birinci sütununa 1 gelmekte olup, bilgisayar bunu sayfa atlama anlamında yorumlar ve her sayfanın başına sadece ikinci sütundaki değerleri yazar.



Çıktı formatının birinci sütununa ASLA format belirteci yazmayınız. Hatalı sayfa düzeni ve hatalı format oluşturma ile sonuçlanabilir!

5.2.2.2 BOŞLUK BIRAKMA—X FORMAT BELİRTECİ

Verilerin gerek girdi gerekse çıktı formatı düzenlenirken boşluk bırakmaksızın yana yana yazılmazlar. İki sayısal değer arasında, okunmasını veya yazmasını veya girdi/çıktı değerlerinin kontrolünde kolaylık sağlaması açısından, belirli bir miktar boşluk konur. **X** format belirtecinin kullanım şekli **nX** veya **nx** olarak verilmektedir. Burada **n** bir satırda bırakılması gereken boşluk sayısıdır; yatay olarak **n** karakter boşluk koyar. Örneğin,

```
INTEGER :: i=15, j=26, K=723
WRITE(6,88) i,j,k
88 FORMAT(1X,I2,9X,I2,5X,I3)
```

Programın çıktısı aşağıdaki şekilde elde edilir:

```
         1         2    2
1234567890123456789012345
 15          26     723
x  xxxxxxxxx  xxxxx
```
→ x boşlukları temsil ediyor

5.2.2.3 SATIR ATLAMA — "/" FORMATI

Verilerin yazdırılması esnasında birkaç satır atlatma veya bir satır alta geçmek ve yazdırma işlemine devam etmek isteyebiliriz. Bu durumlarda "/" format belirteci, format deyimi içinde kullanıldığı yere gelince, yazdırma işlemine devam etmek için bir satır aşağıya geçer. Yukarıdaki örnekte, format deyimi aşağıdaki gibi değiştirilirse,

```
88 FORMAT(1X,i2/2X,i2///3X,i3)
```

bu durumda elde edilen çıktı,

```
         1         2    2
1234567890123456789012345
 15
  26
       boş satır
       boş satır
   723
```

şeklinde olur. Burada dikkat edilecek hususlar şunlardır:

1. Çok sayıda satır atlatmak için **n/** şeklinde bir belirteci KULLANILAMAZ! Örneğin dört satır atlayıp beşinci satırdan başlatmak için **/////** veya **5(/)** şeklinde kullanmalısınız.
2. Satır atlatma belirteci '/', diğer format belirteçlerden virgül ile ayrılması gerekmez. Ancak önüne ve arkasına virgül konabilir.
3. "/" işareti belirttiği satırın birinci sütununu belirler.

5.2.2.4 SÜTUNDAN BAŞLAMA—T FORMAT BELİRTECİ (TAB)

Girdi ve çıktı düzenlerinde, daktilo veya bilgisayar klavyedeki Tab tuşu fonksiyonuna benzer bir işleve sahiptir. Verilerin yazdırılmasında iki veri arasında belirli sayıda boşluk bırakılması için kullanılır. Bu formatın kullanımı **Tn** şeklindedir ve n.ci sütuna gitmeyi belirtir. (Bu esnada kaç boşluk bırakılması gerektiğini belirtmemize gerek yoktur!)

Örneğin, A=l.0, B=2.0, C=3.0 değerleri için

```
REAL :: a=1., b=2., c=3.
WRITE(6,10)
WRITE(6,11) a, b, c
10 FORMAT(2X,'ÇIKTI TABLOSU')
11 FORMAT(T6,'DEĞERLER',/,T10,F3.1,/,T10,F3.1,T16,F3.1)
```

Bu durumda elde edilen çıktı:

```
         1         2    2
1234567890123456789012345
  ÇIKTI TABLOSU
     DEĞERLER
         1.0
         2.0   3.0
```

şeklinde olmaktadır.

5.2.3 TEKRARLANABİLEN FORMAT BELİRLTEÇLERİ

Şu ana kadar bahsedilen format belirteçleri (`/` ve `T` formatı hariç) soldan bir tamsayı ile beraber boşluksuz yazılarak, programa aynı formatın tekrarlandığı bildirilebilir. Örneğin, **`3F6.3`** formatı, bitişik bir şekilde 3 kez `F6.3` formatının yazılmasına, yani **`F6.3,F6.3,F6.3`** formatına eşdeğerdir. Ayrıca parantez içinde belirtilen bir gurup belirteç kombinasyonu da birkaç kez tekrarlanabilir. Örneğin,

```
18 FORMAT(6(/),4X,5(/,3X,I2,2X,F5.2))
```

formatında 6 satır atlandıktan sonra 4 boşluğu takiben `(/,3X,,I2,2X,F5.2)` formatı 5 kez tekrarlanmaktadır. Bu gösterim şekli format deyimlerinin yazılmasında belirgin şekilde kısaltılmalara olanak sağlar.

5.3 METİN GİRDİ/ÇIKTI FORMATI

Programların bir çoğu sadece sayılardan oluşmaz; bu sayılar ile ilgili açıklamalar, etiketler, tablolar v.s. içerebilirler. Metin veya alfa sayısalları sabitler ile değişkenleri de format olarak vermek oldukça kolaydır.

5.3.1 TEK TIRNAK (') VEYA ÇİFT TIRNAK ('') KULLANIMI

Buraya kadar verilen alfa sayısal çıktı örneğinde alfa sayısallar tek veya çift tırnak kullanımı ile kapatılarak aşağıdaki şekilde verilmişti.

```
PRINT*, <alfasayısal ve/veya değişken listesi>
```

Bir alfa sayısal sabit, yine aynı şekilde tek tırnak veya çift tırnak ile kapatılarak ve diğer format belirteçlerden virgül ile ayrılarak `FORMAT` deyimi içinde kullanılabilir. Alfa sayısal sabit, geçerli her tür karakteri (boşluklar dahil) içerebilir ve format deyiminde aynen kullanıldığı şekilde çıktıda gözükürler. Örneğin,

```
INTEGER :: Gun=8, Yil=2005
WRITE(6,20) Gun, Yil
20 FORMAT(5X,'BUGÜN',I3,'   EYLÜL',I6)
```

programının çıktısı

```
          1         2         3
123456789012345678901234567890
     BUGÜN   8 EYLÜL   2005
```

şeklinde elde edilir. Aşağıdaki şekilde verilen format

```
WRITE(6,21)
21 FORMAT(10X,50('*'))
```

10.cu sütundan sonra yan yana 50 adet asteriks (*) ile sonuçlanır.

```
          1                   6
1234567890123456 . . . 4567890
          ****** . . . *******
```



Tek veya çift tırnak işaretleri arasında verilen alfa sayısallarda Türkçe karakterler KULLANILABİLİR!

5.3.2 `WRITE` VE `PRINT` DEYİMLERİNDE (*) YERİNE FORMAT KULLANIMI

`PRINT` ve `WRITE` deyimlerinde format, çok karmaşık olmadığı sürece, asteriks'in bulunduğu konumda verilebilir. Örneğin, `A=2.5`, `B=3.25` ve `I=12` ile `J=47` değerlerinin formatlı olarak çıktısı en basit şekli ile `'*'` yerine format veya format numarası ile yer değiştirilir. Yani,

```
REAL :: A, B
INTEGER :: I, J
CHARACTER(LEN=24) :: form
...
PRINT 35,  A, B, I, J
WRITE(6,35) A, B, I, J
35 FORMAT(5X,2(F7.2,2X),/,5X,2I3)
```

program parçası aşağıdaki şekilde de yazılabilir:

```
form='(5X,2(F7.2,2X),/,5X,2I3)'
PRINT '(5X,2(F7.2,2X),/,5X,2I3)' ,A,B,I,J
WRITE(6,'(5X,2(F7.2,2X),/,5X,213)') A,B,I,J
PRINT form, A,B,I,J
WRITE (6,form) A,B,I,J
```

Burada dikkat edilecek nokta formatın `'` işaretleri ile kapsanması ve parantez içinde yer almasıdır. Ancak formatın çok karmaşık ve uzun olması halinde, asteriks yerine bu formatların verilmesi tavsiye edilmemektedir.

5.3.3 ALFA SAYISAL—`A` FORMAT BELİRTECİ

`CHARACTER` tipinde tanımlanan bir değişkenin çıktısı alfa sayısal format belirteci aracılığıyla sağlanır. Kullanım şekli, **`Aw`** veya **`aw`** olarak tanımlanmıştır. Burada **`w`** karakter ifadenin alan uzunluğudur. Örneğin,

```
CHARACTER(LEN=15) :: isim*12, soyad
isim='AHMET MİTHAT'
soyad='ÖZTÜRK'
WRITE(6,10) isim, soyad
10 FORMAT(5X,'ADI : ',A,'SOYADI : ',A)
```

veya etiket numarası 10 olan formatta A belirteci daha özel olarak

```
10 FORMAT(5X,'ADI : ',A12,'SOYADI : ',A15)
```

şeklinde maksimum alan uzunluklarını içerecek şekilde yazılabilir. Alan uzunluğu A belirteci ile verilmediğinde, alan uzunluğu yazdırılan sabit veya değişkenin CHARACTER ile tanımlanan uzunluğunu alır. Bu bakımdan yukarıda 10 numaralı formatların her ikisinin çıktısı aynı olur.

PRINT deyiminde metin çıktı elde etmek için asteriks yerine aşağıdaki gibi A format belirteci kullanılabilir:

```
PRINT '(A)', <alfasayısal değişken>
```

Örneğin,

```
CHARACTER(LEN=1) :: ASTERISK
ASTERISK= '*'
PRINT '(A)', ASTERISK
```

veya yukarıdaki örnek PRINT deyimi ile aşağıdaki şekilde de ifade edilebilir:

```
CHARACTER(LEN=12) :: isim
CHARACTER(LEN=15) :: soyad
PRINT '(5X,'ADI:',A,'SOYADI: ',A)', isim, soyad
```

5.3.4 ALFA SAYISAL—(:) ŞARTLI DURMA NOKTASI

Yazdırma esnasında format belirteçlerinin davranışını değiştirir. WRITE deyimi için, : sembolü şartlı durma noktası olarak hizmet eder. Eğer yazdırma esnasında iki nokta üst üste işareti ile karşılaşıldığında yazdırılacak başka değerler yoksa, WRITE işlemi durdurulur. Örneğin,

```
REAL, DIMENSION(8) :: x
x = (/ 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8 /)
WRITE (*,100) (i, x(i), i = 1, 8)
100 FORMAT (/,1X,'Çıktı değerleri : '/, &
     3(5X,'X(',I2,') = ',F8.2))
WRITE (*,200) (i, x(i), i = 1, 8)
200 FORMAT (/,1X,'Çıktı değerleri : '/, &
     3(:,5X,'X(',I2,') = ',F8.2))
```

Programının 200 nolu formatında : sembolü kullanıldığında X(9) değeri olmadığı için bu formatın 100 nolu formatta olduğu gibi ‘X(‘ kısmının yazdırılmasını engeller.

```
Çıktı değerleri :
    X( 1) =     1.10     X( 2) =     2.20     X( 3) =     3.30
    X( 4) =     4.40     X( 5) =     5.50     X( 6) =     6.60
    X( 7) =     7.70     X( 8) =     8.80     X(

Çıktı değerleri :
    X( 1) =     1.10     X( 2) =     2.20     X( 3) =     3.30
    X( 4) =     4.40     X( 5) =     5.50     X( 6) =     6.60
    X( 7) =     7.70     X( 8) =     8.80
```

ÖRNEK 2: Bir kişiye ait kişisel bilgileri (adı, soyadı, kilosu, cinsiyeti gibi) ekrandan okuyup ekrana formatlı olarak yazan bir program hazırlayınız.

```
PROGRAM ornek2
IMPLICIT NONE
CHARACTER (LEN=15) :: adi
INTEGER :: yas
REAL :: kilo
CHARACTER (LEN=1) :: cinsiyet
  PRINT *,' Kişinin adını ve soyadını girin '
  READ *, adi
  PRINT *,' Yaşını giriniz '
  READ *, yas
  PRINT *,' kilosunu kg olarak giriniz'
  READ *, kilo
  PRINT *,' cinsiyetini giriniz (E/K)'
  READ *, cinsiyet
  PRINT *,' KİŞİSEL BİLGİLER : '
  PRINT *
  PRINT 100,
  PRINT 200, adi, yas , kilo , cinsiyet
  100 FORMAT(4x,'Adı ve Soyadı',4x,'Yaşı',1x,&
           'Kilosu',2x,'Cinsiyeti')
  200 FORMAT(1x,a,4x,i3,3x,f5.2,2x, a)
END PROGRAM ornek2
```

5.4 READ/WRITE DEYİMLERİNİN DİĞER ÖZELLİKLERİ

Bir girdi kütüğündeki veri satır sayısı genellikle bilinmez. Program bazı verileri okur, hesaplar yapar, READ deyimine döner ve hesapları veri kütüğündeki veriler bitinceye kadar tekrarlar. Buna rağmen, veri kütüğündeki son veri satırından sonra READ deyimi ile okuma yapmaya çalışılırsa bir çalıştırma hatası ile karşılaşılır. Bu nedenle READ deyiminin daha geniş kullanım şekli, bu hataları önlemek için, aşağıdaki şekilde tanımlanmıştır:

```
READ(UNIT=u,FORMAT=format,REC=kayıt,&
  IOSTAT=istat) <Değişken Listesi>
```

burada **IOSTAT**=istat ile bilgisayar bir okuma esnasında hata ile karşılaştığında programa okumanın başarılı veya başarısız olduğunu veya satır veya kütük sonu ile karşılaşıldığını bildirir. Böylece programcı söz konusu durumlarla karşılaşıldığında ne yapılması gerektiği, icabında program akışını, karşılaşılan durumlara göre tedbir almak amacıyla, başka bir yere yönlendirmesine olanak sağlar. REC ise kayıt numarası olup sadece doğrudan (rasgele) erişimli kütüklerde belirli bir kayıtın okunması veya kütüğe yazdırılmasında kullanılır. Bazı örnekler aşağıda verilmektedir:

```
PROGRAM kayitli
IMPLICIT NONE
REAL, DIMENSION(100) :: X, Y, A, B, Z, T
INTEGER :: durum1, durum2, durum3
  OPEN(5,FILE='KAYIT.DAT',ACCESS='DIRECT',RECL=80)
READ(*,40,IOSTAT=durum1) X,Y
  CALL Okuma_Durumu(durum1)
READ(5,40,REC=3,IOSTAT =durum2) A, B
  CALL Okuma_Durumu(durum2)
READ(*,*,IOSTAT=durum3) Z, T
  CALL Okuma_Durumu(durum3)
40 FORMAT(2X,2F5.0)
END PROGRAM kayitli


SUBROUTINE Okuma_Durumu(durum)
IMPLICIT NONE
INTEGER :: durum
IF(durum>0) THEN
    PRINT*, ' ** Veri Okuma Hatası İle Karşılaşıldı **'
ELSE IF(durum=-1) THEN
    PRINT*, ' ** Kütük Sonuna Ulaşıldı **'
ELSE IF(durum=-2) THEN
    PRINT*, ' ** Satır Sonuna Ulaşıldı **'
ENDIF
END SUBROUTINE Okuma_Durumu
```

Bu programda ilk READ deyimi ile X ve Y değerleri 40 numaralı formata göre okunmaktadır, ikincisinde 5 numaralı kütükten 40 numaralı formata göre kayıt numarası 3 olan A ve B değişkenlerini okumaktadır. Son READ deyimi ekrandan ve formatsız olarak Z ve T gibi iki değişkeni okur. Her okumada, okumanın durumu durum değişkeni ile tamsayı olarak programa bildirilir. Okuma_Durumu alt programında durumun aldığı değerlere göre, *okumanın başarılı olması hariç*, uyarı mesajı vermektedir. Bu nedenle alt program her okumadan sonra kullanılmıştır.

Diğer taraftan **WRITE** deyiminin genel yapısı

```
WRITE(UNIT=u,FORMAT=format, REC=kayıt, &
      IOSTAT=istat) <Değişken Listesi>
```

olarak verilmektedir. Anahtar kelime ve belirteçlerin anlamı aynıdır. Çıktı esnasında özellikle format uyuşmazlığı varsa, çalıştırma hatası adını verdiğimiz bir hata türü oluşacaktır. Hatanın nereden kaynaklandığını tespit edebilmek için `IOSTAT` ile hata mesajı oluşturma olanağı mevcuttur. `REC` seçeneği doğrudan erişimli kütüklerde kayıt numarası `REC` ile verilen kayıtın okunması veya ekrana yazdırılmasında kullanılır. Bazı örnekler:

```
WRITE(16,*,REC=5,IOSTAT=durum1)
WRITE(11,45,REC=23,IOSTAT=durum5)
```

olarak verilebilir. Bunlardan ilki 5 numaralı kaydı 16 numaralı kütüğe formatsız olarak yerleştirir, ikincisinde ise 23 numaralı kaydı 11 numaralı kütüğe formatlı olarak yerleştirir.

Girdi/çıktı deyimleri programcıların uzun zamandan beri karşılaştıkları zorlukların kaynaklarından biridir. Kurallar karmaşık, çoğu kez keyfi, ve donanımla sınırlıdırlar. Programcıların yaptığı hataların büyük bir kısmını yanlış girdi/çıktı belirteçleri oluşturur. Yapılan hataların ciddiyeti ve sayısını minimize etmeye yardımcı olmak için bir kaç hatırlatma yapmakta fayda vardır:

- `READ` veya `PRINT` deyimlerinde kullandığınız `FORMAT` deyimlerinin varlığından (programda bir etiket numarası ile tanımlandığından) emin olunuz.
- FORMAT etiket numaralarını sadece *bir kez* kullanınız.
- `READ` veya `PRINT` deyiminde kullanılan değişken listesindeki her bir değişkene karşılık, `FORMAT` deyiminde bir belirteç unsur karşılık gelmelidir. Aşağıdaki örnekte `FORMAT` deyiminde, mesela `F9.3`'ü çıkarırsak, bilgisayar açısından bir belirsizliğe yol açmış oluruz.

```
REAL    :: a, b
INTEGER :: ic, klm
READ 45, a, b, ic, klm
45 FORMAT(f9.3,e14.3,i5,i8)
```

  Yani `a` $\rightarrow$ `f9.3`, `b` $\rightarrow$ `e14.3`, `ic` $\rightarrow$ `i5` ve `klm` $\rightarrow$ `i8` belirteci karşılık gelmektedir.

- Değişken tipi (`REAL`, `INTEGER`, `CHARACTER` v.s) ile `FORMAT`'taki belirteç unsurla uyumlu olmalıdır. Yani tamsayı değişkenler için **`I`** format belirteci, gerçek değişkenler için **`E`**, **`F`** veya **`ES`** format belirteci vb kullanılmalıdır.
- Bir formattaki format belirteçleri ile bunlara karşılık gelen değişken listesi aynı sayıda değil, ancak format tipleri aynı ise, derleyici format tipini kopyalar; bir satıra sığdıramaz ise yeni bir satır açar.

ÖRNEK 3: Bir'den 10'a kadar olan tam sayıların karekökünü, karesini ve küp'ünü hesaplayıp bir tablo halinde veren bir program yazınız.

```
PROGRAM Ornek3
IMPLICIT NONE
INTEGER :: kup          ! i sayısının kübü
INTEGER :: i            ! İndis değişkeni
INTEGER :: kare         ! i sayısının karesi
REAL    :: kare_kok     ! i sayısının kare kökü
```

```
! Yeni sayfaya tablonun başlığını yaz
WRITE (*,100)
100 FORMAT ('1', T3, 'Karekök, kare ve küp Tablosu')

! Sütun başlıklarını yaz
WRITE (*,110)
110 FORMAT ('0',T4,'Sayı',T13,'Kare Kök',T29, &
   'Kare',T39,'Küp')
WRITE (*,120)
120 FORMAT(1X,T4,'======', T13, &
   '===========',T29,'======',T39,'===='/)

! Arzulanan değerleri hesapla ve ekrana yaz
DO i = 1, 10
   kare_kok = SQRT(REAL(i))
       kare = i**2
       kup  = i**3
       WRITE(*,130) i, kare_kok, kare, kup
END DO
130 FORMAT (T4, I4, T13, F10.6, T27, I6, T37, I6)
END PROGRAM Ornek3
```

Programın çıktısı aşağıdaki şekilde gerçekleşir:

```
Karekök, kare ve küp Tablosu
Sayı     Karekök         Kare      Küp
======   ===========     ======    ====

   1       1.000000         1         1
   2       1.414214         4         8
   3       1.732051         9        27
   4       2.000000        16        64
   5       2.236068        25       125
   6       2.449490        36       216
   7       2.645751        49       343
   8       2.828427        64       512
   9       3.000000        81       729
  10       3.162278       100      1000
```

ÖRNEK 4: Bir'den 10'a kadar olan tam sayıların karekökünü ve küp kökünü hesaplayıp bir tablo halinde veren bir program yazınız.

```
PROGRAM Ornek4
IMPLICIT NONE
INTEGER, PARAMETER :: max_boyut = 10
INTEGER :: j
REAL, DIMENSION(max_boyut) :: deger
REAL, DIMENSION(max_boyut) :: kare_kok
REAL, DIMENSION(max_boyut) :: kup_kok
```

```
! Değerlerin karekök ve küp köklerini hesapla.
DO j = 1, max_boyut
   deger(j) = real(j)
   kare_kok(j) = sqrt(deger(j))
   kup_kok(j)  = deger(j)**(1./3.)
END DO
! Her sayıyı, kare kökünü ve küp kökünü ekrana yaz.
WRITE (*,100)
WRITE (3,110) (deger(j), kare_kok(j), kup_kok(j),&
             j = 1, max_boyut)
100 FORMAT('0',20X,'Karekök ve küp kök Tablosu',/, &
    4X,'  Sayı      Karekök      Küp Kök  ',  &
    3X,'  Sayı      Karekök      Küp Kök  ',/,&
    4X,'  ======    ===========  =========',  &
    3X,'  ======    ===========  =========')
110 FORMAT (2(4X,F6.0,9X,F6.4,6X,F6.4))
END PROGRAM Ornek4
```

Programının çıktısı aşağıdaki düzende gerçekleşmektedir:

```
                Karekök ve küp kök Tablosu
Sayı     Karekök     Küp Kök     Sayı     Karekök    Küp Kök
======   ==========  =========   ======   ========== =========
  1.       1.0000     1.0000       2.      1.4142     1.2599
  3.       1.7321     1.4422       4.      2.0000     1.5874
  5.       2.2361     1.7100       6.      2.4495     1.8171
  7.       2.6458     1.9129       8.      2.8284     2.0000
  9.       3.0000     2.0801      10.      3.1623     2.1544
```

ALIŞTIRMALAR

5.1 Aşağıdaki deyimlerde varsa hatalı olanları hataları ile beraber tespit ediniz.

```
a)  READ*; X,Y,Z          b)   PRINT*, X;Y;Z
c)  READ(5,*) X2,Y3       d)   READ(*) A,AX
e)  READ*, X+Y+Z          f)   READ(6,5),X,Y
g)  WRITE(*,) X,Y         h)   WRITE(*,3) X;Y
i)  WRITE(*,*) X+Y,Z      j)   PRINT*, X,Y,X+Y
k)  WRITE(3,3) X**2       1)   WRITE(*,5) 'X=';X
m)  PRINT*, 'X+Y',X+Y     n)   PRINT 15,A,B,X,I
```

5.2 Aşağıdaki deyimlerde varsa hataları tespit ediniz.

```
a) READ(5,10) N,M,L      b) READ(5,11) X,N,L,Y
   10 FORMAT(I3,I5,3X,I4)   11 FORMAT(F5.l,F4,I3,F10.4)
c) READ(5,12) X,Y,M,N    d) READ(5,13) X,N,L,Y
   12 FORMAT(5X,2F10.3,2I4) 13 FORMAT(2X,3(F5.2,IX),15)
e) READ(5,14) X,Y,L,Z    f) READ(5,15) X,Y,Z,T
   14 FORMAT(2E10.2,I10.2)  15 FORMAT(5(1X,E12.5))
g) READ(5,16) U,IU,Z,NZ  h) READ(5,17) A,B,IA,IB
   16 FORMAT(2(F6.0,1X,I3)) 17 FORMAT(1X,2F5.0,IX,215)
```

5.3 `READ` listesinde uygun değişkenlerin denk geldiğini varsayarak aşağıda verilen `FORMAT` deyimlerinde varsa hatalı olanlarını bulunuz.

```
(a)   10 FORMAT(1X,I2,F3.0,I2,F3.0)
(b)   15 FORMAT(2X,I3,f3.1,2X,F5.1,3X,f7.4)
(c)   20 FORMAT(5X,I4,e14.4,5x,I3,2X,F6.2)
(d)   25 FORMAT(2X,I8,1X,F7.2E+02,F4.2)
(e)   30 FORMAT(T20,I5,T25,I5,T30,F5.1,T40,F5.2)
(f)   35 FORMAT(//,2(I5,2/,F4.2))
(g)   40 FORMAT(3X,3(I2,F3.1,I2))
(h)   45 FORMAT(x,5F3.5,4I3)
(i)   50 FORMAT(T15, 2(15, E7.1),2x, 3(14, E8.0))
(j)   55 FORMAT(10X,3(2X,f6.2,I7)
(k)   60 FORMAT(T10,4E5.3,f12.3)
```

5.4 Aşağıda verilen `WRITE` deyimlerinde varsa hatalı olanları tespit ediniz.

```
a)   WRITE(6,10) A, B, C
     10 FORMAT(4f5.1)

b)   WRITE(6,11) X,X*X,X**3
     11 FORMAT(5E12.4)

c)   WRITE(6,12) X,L,Y,K
     12 FORMAT(1X,3('X=',e9.2,"K=",17))

d)   WRITE(6,13) X,Y,Z,T,U
     13 FORMAT(1X,45('-'),/,1X,45('='),/,1X, &
          45('-'),7(1x,e8.1))

e)   WRITE(6,14) X,'Y=',Y
     14 FORMAT(2X,'X=',F7.3,A2,E11.4)

f)   WRITE(6,15) "X=",X, 'Y=',Y
     15 FORMAT(5X,A2,F6.3,/,5X,A2,E11.4)

g)   WRITE(6,16) X,Y,Z
     16 FORMAT(///,4X,"X İN DEĞERI=",e8.l,/,3F9.3/7)
```

5.5 Aşağıdaki deyimlerde `X=4.`, `Y=3.`, `Z=l./6.`, `I=4`, `J=3`, `K=2` değerlerini aldıklarını varsayarak çıktıları bulunuz.

```
a)   WRITE(6,10) X,Y,Z
     10 FORMAT(2X,F3.1,2X,F2.0,2X,F7.4)

b)   WRITE(6,11) X,Y,Z
     11 FORMAT(/,2X,F7.5,3(/,2X,f6.3,/))

c)   WRITE(6,12) X,I,I*I
     12 FORMAT(5X,F3.1,'*',I2,'=',i2)

d)   WRITE(6,13) X*Z,Y,K,J,J
     13 FORMAT(/,4X,f9.6,'*',F6.2,'=',i4,/,3i3)

e)   WRITE(6,14) K,I,J
     14 FORMAT(t12,i1,T22,i1,T28,i1)
```

```
f)   WRITE(6,15) I,J,K
     15 FORMAT(T2,I1,I5,I1,T7,I1)
g)   WRITE(6,16) I,J,K
     16 FORMAT(5X,I2)
h)   WRITE(6,17) X,Y,Z,I,J,K
     17 FORMAT(5X,3F8.5,3X,3I4)
i)   WRITE(6,18) X,I,Y,J,Z,K
     18 FORMAT(2X,4(e9.3,2x,i3,1X))
```

5.6 Bir'den 25'e kadar olan tamsayıları, karelerini, küplerini ve kareköklerini hesaplayıp, ekrana yazan bir program yazınız. NOT: Çıktının aşağıdaki formatta olması istenmektedir.

```
         1         2         3         4         5
12345678901234567890123456789012345678901234567890
         SAYI      KARESİ      KÜBÜ        KAREKÖKÜ
           1          1          1         1.00000
           2          4          8         1.41421
          ..         ...        ...          ...
```

5.7 Aşağıdaki deyimlerde X=3., Y=5., Z=l./3., I=3, J=2, K=4 değerlerini aldıklarım varsayarak çıktıları bulunuz.

```
(a)  PRINT '(f4,0,2x,f4.1,f3.1,2i3)',X,Y,Z,I,K
(b)  WRITE(6,10)  X,Y,I,J
     10 FORMAT(T5,2e9.1,/,2i2)
(c)  WRITE(*,'(F5.2,a4)') X,'=X'
(d)  WRITE(*,'(F6.3,A3,I5)') X,'***',I,Y,'---',J
(e)  WRITE(6,11)  X,1./X,1./X**2,I*Z-1.
     11 FORMAT(5(1x,e10.3))
(f)  WRITE(6,12)
     12 FORMAT(2X,5('*',2X),3('-'))
(g)  WRITE(6,13)  X, Y, Z, X+Y+Z
     13 FORMAT(5X,'X=',F5.2,'Y=',F5.2,*Z=',F5.2,/,&
               1X,'X+Y+Z=',F8.5)
(h) WRITE(*,'(1X,F5.1,/,1X,I2)') X,I,Y,J,Z,K
```

5.8 Aşağıdaki çıktıyı elde etmemizi sağlayacak sadece tek FORMAT deyimi ile hazırlayınız.

```
      AAAAA
      AA  AA
     AA    AA
    AAAAAAAAA
   AA        AA
  AA          AA
AAAAA        AAAAA
```

5.9 Herhangi bir harfi yine kendi harfleri kullanılarak, (5.8).ci problemde verilen ebatta yazdıran bir program yazınız.

5.10 Bir eczanedeki 235 tür ilaca sırasıyla l'den 235'e kadar numaradan oluşan kod'lar verilmektedir. Bu ilaçlar, sırasıyla `Ilac_Kodu` (**`I3`**), `ilac_ismi` (**`A20`**), `ilac_fiyati` (**`F9.0`**) olarak ECZA.DAT kütüğünde saklı tutulmaktadır. Bu ilaçlardan kodu 1-90 arası olanlar *Vitamin*, 91-135 arası olanlar *Antibiyotik*, 136-159 arası olanlar *Antihistamin*, 160-192 arası olanlar *Ağrı kesici ve Ateş düşürücüler*, 193 ile 235 arası olanlar *yara pomadları* oluşturmaktadır. İlaç firmalarınca aynı hafta yapılan açıklamalar ile vitaminlere %13, Antibiyotiklere %22, Ağrı kesicilere %18 zam yapıldığını belirtilmektedir. Eczacı yeni fiyat listesini %15 KDV'yi dahil ederek hazırlamak için sizi tuttuğunu farzedersek, bu eczacının işini görecek çıktısı aşağıdaki formatta olan bir programı nasıl yazarsınız? (`x`'ler sayının işgal edeceği yazdırma alanını göstermektedir)

```
         1         2         3         4         5         6
123456789012345678901234567890123456789012345678901234567890
 Kodu  İsmi                Fiyatı      Kdv          TOPLAM
 xxx   xxxxxxxxxxxxxxxxxxxx  xxx.xx     xxx.xx      xxxx.xx
```

5.11 Bir bankanın Genel Müdürlüğünün tüketici kredisi veren birimine yapılan başvurular- dan şu bilgiler değerlendirmeye alınıyor: `ISIM` (başvuru sahibinin adı ve soyadı, `A40`), `Kredi` (istediği kredi miktarı, `F7.3`), `Gelir` (aylık geliri, `F8.2`), `YIL` (geri ödeme süresi, `i2`) gibi bilgiler DATA.DAT kütüğünden okunacaktır. Birimin istediği program, başvuru sahibine kredinin verilmesi durumunda geri ödemesi gereken taksit miktarını hesaplayacak; bu miktar aylık gelirinin %55'in altında ise, müşteri kredi almaya hak kazanacaktır. Bu durumda ekrana `'Tebrikler!!! Kredi almaya hak kazandınız'` aksi durumda `'Malesef size kredi veremiyoruz'` yazacaktır. Aylık taksitler aşağıdaki formülle hesaplanmaktadır. Aylık faizi % 5.75 olarak alınız.

```
Taksit=kredi * r * ((1.+r)**sure-1.)/(1.+r)**sure
```

Burada `r=Faiz/100`; `Faiz`=Aylık faiz, `sure`=kredinin geri ödeme süresi (ay olarak) =`12*YIL`. Kredi almaya hak kazanan müşteri için, diğer bilgileri içeren, bilgisayar çıktısının aşağıdaki gibi olması istenmektedir:

```
         1         2         3         4         5         6
123456789012345678901234567890123456789012345678901234567890
     ADI VE SOYADI : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
     GELİR         : xxxxx.xx YTL
     KREDİ MİKTARI :  xxxx.xx YTL   YIL : xx   FAİZ : % x.xx
 TAKSİT        :  xxxx.xx YTL
```

5.12 Türkiye Elektrik Kurumunda (TEK) çalışan bir mühendisten hazırlanması istenen bir program ile ilgili sorular ve veriler şunlardır: Kurumdaki abonelerin isim ve soyadları `ad_ve_soyad` (**`A50`**), adresleri `adres` (**`A75`**), abonenin statüsü (`statu_kod`=statü kodu, Mesken için `statu_kod`=0, işyeri için `statu_kod`=1, **`I1`** formatında), tüketilen elektrik enerjisi `tuketim_KWH` (`F9.2`). Bu bilgiler her ay bilgisayara girilmektedir. Kurum, enerjinin kilowatt saatini meskenlere 0.250 YTL'ına, işyerlerine ise 0.225 TL'ına satmaktadır. Mesken abonelerinin aylık tüketimi 120 kwh, işyeri abonelerinin aylık tüketimi 265 kwh aştığında, aşan miktara uyguladığı tarife meskenlere %18,

işyerlerine %13 zamlıdır. Hesaplanan ücrete %15 KDV ayrıca ilave edilecektir. Faturaya abonenin ismi ve adresi, statüsü, tüketim, tüketim ücreti, KDV ve ödenmesi gereken toplam ücret (fiyat+KDV) yazılması isteniyor.

5.13 Bir fizik öğretmeni uyguladığı yazılı ve sözlü sınavlardan A, B, C, D ve F notları vermektedir. Bu notlara karşılık gelen sayısal ağırlıklar sırasıyla 100, 75, 50, 25 ve 0'dır. Öğretmen bir dönemde 3 yazılı ve 2 sözlü sınavı uygulamaktadır. Dönem sonu not ortalaması hesaplandıktan sonra ortalama değerin en yakın olduğu harf seçilerek dönem ortalaması verilmektedir. M kişiden oluşan bir sınıfın dönem sonu notunu harfli sisteme göre hesaplayan bir program yazınız. Sınav notlarının da harf olarak girildiğini kabul ediniz.Örneğin, yazılıları B, B, A ve sözlüleri A, C olan bir öğrencinin ortalama notu

$$\frac{(75+75+100)+(100+50)}{5}=80$$

olarak bulunur; bu not ortalama 75'e daha yakın olduğundan dönem ortalaması B seçilir.

5.14 Bir ilin emniyet teşkilatında sabıkalılara ait bir veri tabanı oluşturulmaktadır. Bu veri tabanında sabıkalılara ait adı ve soyadı (`ad_ve_soyad`, **`A40`**), adresi (`adres`, **`A40`**), yaşı (`yas`, **`I2`**), boyu (`Boy`, **`F5.2`**), cinsiyeti (`cinsiyet`, **`Al`**, seçenekler `'E'`,`'K'`), medeni hali (`Medeni_Hal`, **`I1`**, `Medeni_Hal=0` Bekar, `Medeni_Hal=1` Evli, `Medeni_Hal=2` Dul, `Medeni_Hal=3` Boşanmış), tipi (`TIP`, **`A7`** seçenekler `'ESMER'`, `'SARIŞIN'`, `'KUMRAL'`), saç rengi (`Sac_Rengi`, **`A5`**, `'KAHVE'`,`'SARF'`,`'SIYAH'`), göz rengi (`Goz_Rengi`, **`A5`**, `'ELA'`, `'KAHVE'`, `'MAVİ'`,`'YEŞİL'`) ve suç cinsi (`Suc_Cinsi`, hırsızlık=0, dolandırıcılık=l, cinayet=2, gasp=3 v.s gibi) bilgiler olacaktır. Bir istatistik gereği; (a) yaşları 18-25, boyu 1.70-1.78 m arasında olan erkek sabıkalıların sayısını ve listesini çıkartan program, (b) yaşları 23-29, boyu 1.63-1.72 m arasında olan evli ve hırsızlık yapan kişilerin sayısını veren programı yazınız. Program çıktısı aşağıdaki şekilde olması istenmektedir.

```
         1         2         3         4         5         6
123456789012345678901234567890123456789012345678901234567890
    YAŞLARI 18-25 ARASINDA, BOYU  1.70-1.78 m ARASINDA OLAN
    SABIKALILAR xxx KIŞIDIR. LİSTE AŞAĞIDA VERİLMİŞTİR.

  *** xx.   ci SABIKALININ ***
  ADI SOYADI   : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  ADRESİ       : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  YAŞ          : xx     BOY : xx.xx m    CİNSİYET : x
  MEDENİ HAL   : xxxxx  TIP : xxxxxxx    GÖZ RENGİ: xxxxx
  SAÇ RENGİ    : xxxxx  SUÇ CİNSİ : xxxxxxxxxxxxxx
```

5.15 `N` kişiden oluşan bir sınıftaki öğrencilerin notları `z`; ve not ortalaması da `z` ile temsil edilmektedir. Bu sınıf için standart sapmayı (`s`'yi) hesapladıktan sonra öğrencilerin notlarına göre aşağıdaki şekilde bir gruplandırma yapılmak istenmektedir

Not Aralığı	Durum
$z-s \le x_i < z$	ORTA
$z \le x_i < z+s$	İYİ
$z+s \le x_i < z+1.5s$	ÇOK İYİ

```
         1         2         3         4         5         6
123456789012345678901234567890123456789012345678901234567890
    ADI            SOYADI      NOT     DURUM
 ==================================================
    Ahmet          TÜRK         67     İYİ
    Mehmet         ÖZTÜRK       90     ÇOK İYİ
    Ayşe           DEMİR        53     ORTA
    . . .           . . .       . .    . .
 ==================================================
```

5.16 Yukarıdaki soruda **(5.15.** soru) notu 0-19, 20-39, 40-59,... ve 80-100 sınırları arasında olan öğrencilerin sayısını (frekansını) veren bir program yazınız. Bilgisayar çıktısı aşağıdaki gibi olsun

```
         1         2         3         4         5         6
123456789012345678901234567890123456789012345678901234567890
============================================================
!   xxxxxxxxxxx  Dersi İstatistikleri                      !
!                                                          !
!    SINIF MEVCUDU : xxx                                   !
!    SINAVA GİRENLERİN SAYISI     : xxx                    !
!    SINAVA GİRMEYENLERİN SAYISI  : xxx                    !
!    SINIF ORTALAMASI : xx.x                               !
!    DAĞILIM FREKANSI                                      !
!    ----------------                                      !
!     0-19    xx kişi                                      !
!    20-39    xx kişi                                      !
!       . . .                                              !
!    80-100   xx kişi                                      !
============================================================
```

BÖLÜM 6
İNDİSLİ DEĞİŞKENLER VE DÖNGÜLER

Bu bölüme kadar verilen FORTRAN deyimleri birçok problemin çözümü için yeterlidir. Buna rağmen, bazı problemlerde Fortran'ın diğer özelliklerini tanımadan kolaylıkla çözmek oldukça zordur. Örneğin, 100 kişiden oluşan bir sınıftaki öğrencilerin not ortalamasını ve standart sapmasını bulmak istersek, programlanması oldukça zor olan bir problem ile karşı karşıya kalırız.

Bu problemde öğrencilerin notunu, hem ortalamayı hem de standart sapmayı hesaplarken kullanmak gerekmektedir. Öğrencilerin notlarını programa

```
REAL :: Ogrenci1, Ogrenci2, ..., Ogrenci100
READ(65,*) Ogrenci1, Ogrenci2, ..., Ogrenci100
Ortalama=(Ogrenci1 + Ogrenci2 +...+ Ogrenci100)/100.
Sapma=SQRT(( (Ogrenci1-Ortalama)**2 +...+ &
       (Ogrenci100-Ortalama)**2 )/99.)
```

şeklinde okuyabilir; ortalama ile standart sapmayı hesaplayabiliriz. Bu işlemleri uygulayarak söz konusu hesapları kodlamak sadece uzun zaman almakla kalmaz, aynı zamanda program tarzını çirkinleştirir. Buna bir de programda kullanılması gereken veri sayısı 100'ler ile değil de binler ve yüz binlerle ifade etmeye kalktığımızda yaşayacağımız sorunlar daha da çoğalır. Matematikte bunca sayıyı indisli olarak ifade ederiz. Bu ifade şekli de programlama dillerine uyarlanmıştır. Ortalamanın cebirsel ifadesinden hareketle,

$$\text{Ortalama} = \frac{1}{100}\sum_{i=1}^{100}\text{Ogrenci}(i)$$

öğrenci notlarını indisli olarak tanımladığımızda $Ogrenci_1$, $Ogrenci_2$,…, $Ogrenci_{100}$ yazmak ile problem oldukça basitleşir. İndisli değişken tanımı tip tanımlama kısmında `DIMENSION` deyimi ile gerçekleştirilir.

6.1 DIMENSION DEYİMİ

Daha önce bahsedildiği gibi, derleyici bir değişken ile karşılaştığında, bu değişken için bellekte, tipine göre, bir alan ayırır. Bellekteki bu alanın içeriğine ulaşmak için değişken isminin kullanılması yeterlidir. Derleyiciye bir değişken için birden fazla bellek alanının ayrılması talimatı verilebilir. Böylece bir değişken (indisli olmak şartıyla) birden fazla belleğe ama aynı değişken ismine sahip olur; yalnız doğal olarak her bellek alanının adresi farklıdır.

İndisli (veya boyutlu) değişken tanımlamak için kullanılan FORTRAN deyimi `DIMENSION` deyimidir. Programda değişken tiplerinin tanımlandığı kısımda, aşağıdaki şekilde tip tanımlama ile beraber kullanılır.

```
INTEGER, DIMENSION(100) :: ogrenci
```

```
REAL, DIMENSION(10,15)  :: a
```

Burada `ogrenci` değişkeni 100 adet "tamsayı"yı [`ogrenci(1), ogrenci(2),. . .,` `ogrenci(100)` olarak] ve `a` değişkeni ise 10×15 adet "gerçek sayı"yı [`a[1,1], a[1,2],..,a(1,15),a(2,1),a(2,2),. .,a(2,15),. .,a(15,1),. ., a(10,15)`]bellekte saklayabilecek alanlar oluşturmaktadır. `Ogrenci` indisli değişkeni için oluşturulan bellek ve adresleri Şekil 6.1'de şematik olarak gösterilmektedir. Her bellek adresine sadece bir adet "tamsayı" yerleştirilebilmektedir. Örneğin, 71.inci adresteki değere ulaşmak ve bu değeri kullanmak için `ogrenci(71)` şeklinde indisli değişkenin adresini parantez içinde belirtmemiz yeterlidir.

1	2	3	...	98	99	100

Şekil 6.1 Bir boyutlu indisli değişkenler için bellek ayırımı.

Tek indisli değişkenlerin bellek alanlarını bir şeride benzetebiliriz. İki indisli değişkenlerde de, Şekil 6.2'de görüldüğü gibi, iki boyutlu (matrislere benzer) şekilde bellek adresi oluşturulur. Aslında bu bellek adresleri bizim daha iyi anlamamız için bu şekilde bir benzetme kullanılmaktadır; oysa iki ve daha fazla indisli (boyutlu) değişkenlerin gerçek bellek alanları Şekil 6.1'deki gibi şerit halindedir. Bu kısma ileride değinilecektir.

i,j   j →

i ↓

1,1	1,2	1,3
2,1	2,2	2,3
3,1	3,2	3,3
4,1	4,2	4,3
5,1	5,2	5,3

Şekil 6.2 İki boyutlu indisli değişkenler için bellek ayırımı.

İndisli değişkenleri tamsayı, gerçek, mantıksal veya alfa sayısal v.b tiplerde oluşturabiliriz. İndisli olması istenilen değişkenler `DIMENSION` deyimini takiben parantez içinde ayrılması gereken hafıza alanı belirtilmeli veya değişkenin alması arzulanan indis aralığını vermelidir.

Genel kullanım şekli

```
Tip, DIMENSION(boyut_ve_alan) :: <Aynı bellek alanına
                      sahip değişkenlerin listesi>
```

olarak verilmektedir. Burada `boyut_ve_alan` indisli değişkenin boyutunu virgül ile ayırarak, her indisin bellekte alabileceği üst değeri belirtmekte kullanılır.

Bazı örnekler:

```
REAL, DIMENSION(2,3,1) :: gec, alan
CHARACTER(LEN=3), DIMENSION(89) :: cevap
REAL, DIMENSION(100)   :: Ogrenci
REAL, DIMENSION(5,6)   :: A
INTEGER, DIMENSION(16) :: Kod
REAL, DIMENSION(5,5,5,5,5,5) :: Tempo
```

Yukarıda `gec` ve `alan` değişkenleri üç boyutlu 2×3×1'lik "gerçek" sayılardan oluşan indisli değişken olarak tanımlanmıştır. Birinci, ikinci ve üçüncü indisler, sırasıyla 1'den başlayarak en fazla 2, 3 ve 1 değerlerini alabilirler. `Cevap` karakter uzunluğu 3 olan alfa sayısal değişkenine 89 adet bellek alanı (1'den 89'a kadar) ayrılmıştır. **`DIMENSION`** deyimi ile bir boyutlu/indisli `Ogrenci` değişkeni için bellekte 100 gerçek sayı alanı oluşturulur. Bellekteki ilk alan `Ogrenci(l)` ve sonuncusu `Ogrenci(100)` olarak değerlendirilir. Parantez içindeki sayı, dolayısıyla, pozitif bir tamsayı olmalıdır. Gerçek bir değişken olan `A` iki boyutlu/indisli bir değişken olup, 30 bellek alanı içermektedir; yani $a_{i,j}$ şeklindeki değişkenin $i = l\text{-}5$ ve $j = l\text{-}6$ (alan=5×6=30) arasında değişir. `Kod` tamsayı değişkeni bir boyutludur ve 1'den 16'a kadar adreslenen 16 bellek alanı ayrılmıştır. Altı indise sahip `Tempo` değişkeninin her indisi 1'den 5'e kadar değerler almaktadır; oluşturulan toplam bellek alanı 5×5×5×5×5×5=15625'tir.



FORTRAN'da bir indisli değişken en fazla 7 indise sahip olabilir.

6.1.1 TİP VE `DIMENSION` DEYİMİNİN BİRLEŞTİRİLMESİ

FORTRAN'da tip tanımlama ve `DIMENSION` deyimi aşağıdaki şekilde de kullanılmaktadır. Ancak FORTRAN 90/95 dillerinde bu iki deyim tek deyim altında birleştirilmektedir.

```
REAL :: KAT
DIMENSION KAT(18)
```

deyimi

```
REAL, DIMENSION(18) :: KAT
```

şeklinde birleştirilir. Tamsayı, mantıksal, kompleks veya alfa sayısal olan indisli değişkenler de benzer şekilde tanımlanabilir.

```
CHARACTER(LEN=10), DIMENSION(49):: C
REAL, DIMENSION(999)   :: Vergi, Maas, KDV
COMPLEX, DIMENSION(55) :: SanalZ1, SanalZ2
LOGICAL, DIMENSION(200):: Evet_Hayir
```



Tip tanımı ile `DIMENSION` *tanımını ayrı ayrı vermeyiniz. Bu özellik FORTRAN 77'den kalma bir özelliktir. FORTRAN 90/95'de* `DIMENSION` *ve "Tip" belirleme deyimlerini birlikte kullanınız.*

6.1.2 DIMENSION DEYİMİNDEKİ İNDİSİN ALT VE ÜST SINIRLARI

Tek indisli bir değişkenle `DIMENSION` deyiminin en genel kullanımı

```
Tip, DIMENSION(alt_indis:üst_indis) :: <indisli değişken>
```

olarak verilir. Burada `alt_indis` indisli değişkenin alt sınırı, `üst_indis` indisli değişkenin üst sınırını temsil etmektedir. Bu deyimi aynı indis sınırlarına sahip değişken listesi takip eder. Alt indis sınırı verilmediğinde, bu sınır derleyici tarafından `1` olarak algılanır. Örneğin,

```
REAL, DIMENSION(15) :: X
INTEGER, DIMENSION(-3:12):: cevap
REAL, DIMENSION(3:23) :: maas
REAL, DIMENSION(-3:7) :: c
```

şeklinde verilmiş olsun. Burada X_n değişkeninde indis *n*=`l` ile *n*=`15` arasında değişir. *`cevap`* indisli değişkeni *n* = `-3` ile *n* =`12`, `maas` indisli değişkeninin indisi de *n* = `3` ile *n* = `23` arasında değişmektedir. `c` değişkeninde ise indisleri `-3`'den `7`'e değişen, `c(-3)`, `c(-2)`, . . ., `c(7)` şeklinde, "gerçek" sayılardan oluşmaktadır.

Diğer taraftan, iki veya daha fazla indisli değişkenlerde her indis aralığı virgül (`,`) ile ayrılır ve benzer şekilde indis aralıklarının alt ve üst sınırları `:` ile belirlenir. İki indisli bir değişken için genel kullanım şekli aşağıda verilmektedir.

```
Tip, DIMENSION(alt1:üst1, alt2:üst2) :: <indisli değişken>
```

Örneğin,

```
REAL, DIMENSION(5,7)      :: A
REAL, DIMENSION(-1:3,0:4) :: indisli
```

tanımlarında `A` iki indisli değişken olup, değişken a_{ij} şeklinde yazılabilir. Burada da alt indisler verilmediğinden `1` olarak kabul edilir ve indisler *i* = `l`'den `5`'e ve *j*=`l`'den `7`'ye kadar değişir. Diğer taraftan `indisli` değişkeninin iki indisi olup, bunlardan `i`, `-l` ile `3` ve `j`'nin `0` ile `4` arasında değiştiği bildirilmektedir.

İndisli değişkenlerin indisleri aşağıdaki örnekte olduğu gibi *değişken atama* yapılamaz.

```
INTEGER :: N
READ*, N
REAL, DIMENSION(N) :: A          !  Yanlış
REAL, DIMENSION(N+1,2*N) :: B    !  Yalnış
```

Bu geçerli bir tanımlama değildir. Yalnız **PARAMETER** niteliği kullanılarak, aşağıdaki gibi, değişken görünümünde kullanılabilir. Burada PARAMETER deyimi ile atanan N değeri program boyunca sabit kalır; değiştirilemez. Aşağıdaki tanımlama *geçerli* bir tanımlamadır.

```
INTEGER, PARAMETER :: N=5
REAL, DIMENSION(N)       :: AMAT
REAL, DIMENSION(2*N)     :: BMAT
REAL, DIMENSION(N+1,2*N) :: CMAT
```

veya

```
INTEGER, PARAMETER           :: alt=10, ust=10
REAL, DIMENSION(alt, ust)    :: amat
REAL, DIMENSION(2*alt, 3*ust):: bmat
REAL, DIMENSION(alt+2,ust+5) :: cmat
```

şeklinde tanımlama yapılabilir.

Tip tanımlama kısmında **PARAMETER** niteliği ile çeşitli sabitlerin program boyunca değeri sabit kalır ve bu değeri değiştirilemez.

```
INTEGER, PARAMETER :: N=5
INTEGER :: M=5
N=N+2   ! Müsaade edilmez çünkü programda N daima 5 olarak kalacaktır
M=M+2   ! Müsaade edilir; çünkü M=5 ilk bellek yani başlangıç değeridir
```



İndisli değişkenlerin alt ve üst sınırlarının değişken olarak tanımlanabilmesi için tip ve DIMENSION *tanımından önce değişkenin* **PARAMETER** *ile verilmesi gerekir.*

6.1.3 İNDİSLİ DEĞİŞKENLERİN BELLEKTE DEPOLANMASI

Program derlendikten sonra, indisli değişken boyutu ne olursa olsun, tamamı bellekte tek sıra olarak saklanır. Dolayısıyla, A(4) şeklinde verilen bir indisli değişkenin aldığı değerler bellekte sırasıyla a_1, a_2, a_3, a_4 şeklinde depolanır. İki indisli A(4,4) değişkeninin ilk indisi satır, ikinci indisi sütunu temsil eden bir matris olarak düşünülebilir.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix}$$

Bilgisayar bu değerleri sıralı olarak saklandığından dolayı, bu matrisin değerleri bellekte sırasıyla, Şekil 6.1-b de verildiği gibi, $a_{1,1}, a_{1,2}, a_{1,3}, \ldots, a_{2,1}, a_{2,2}, \ldots$ olarak (yani sütunlar arka arkaya ilave edilir-Şekil 6.1-a) depolar.

Bir indisli değişken `A(L1,L2)` olarak (buradaki `Ll` ve `L2` sabit tamsayı değerlerdir) boyutlandırılmışsa, bunun herhangi bir `i`, `j` elemanın, tek sıralı bellek adresi

n=i+(j-1)*L1

ile verilir. Örneğin, yukarıda verilen A matrisinin (2,3)'deki elemanı n=2+(3-1)*4=10'dur; yani değeri onuncu bellek alanında yer almaktadır. Bu saklama şeklinin girdi ve çıktı ifadelerinde önemli sonuçları vardır.

```
REAL, DIMENSION(5,7) :: A
WRITE(*,*) A
```

şeklinde verilen bir FORTRAN deyiminde A matrisinin değerlerinin ekrana yukarıda belirtilen sırayla formatlı veya formatsız olarak yazılmasına neden olur. Benzer şekilde,

```
REAL, DIMENSION(5,7) :: A
READ(*,*) A
```

ifadesinde A'nın 35 değeri yukarıda belirtilen bellek adresi sırasıyla konsol veya kütükten okur.

Üç indisli değişkenler söz konusu olduğunda, `A(2,4,3)` gibi, depolama sırası düzlemler boyunca olur; her düzlem bir matris gibi düşünülerek bu düzlemdeki sıralama matris sıralaması gibi yapılır; sonra diğer düzleme geçilerek bu işlem devam ettirilir (Şekil 6.3-c). Üçüncü indisin maksimum değeri, buradaki A değişkeninde 3 tür, düzlem sayısını verir.

Herhangi bir `A(L1, L2, L3)` üç indisli değişkenin herhangi bir `i,j,k`'ıncı elemanının bellek adresi

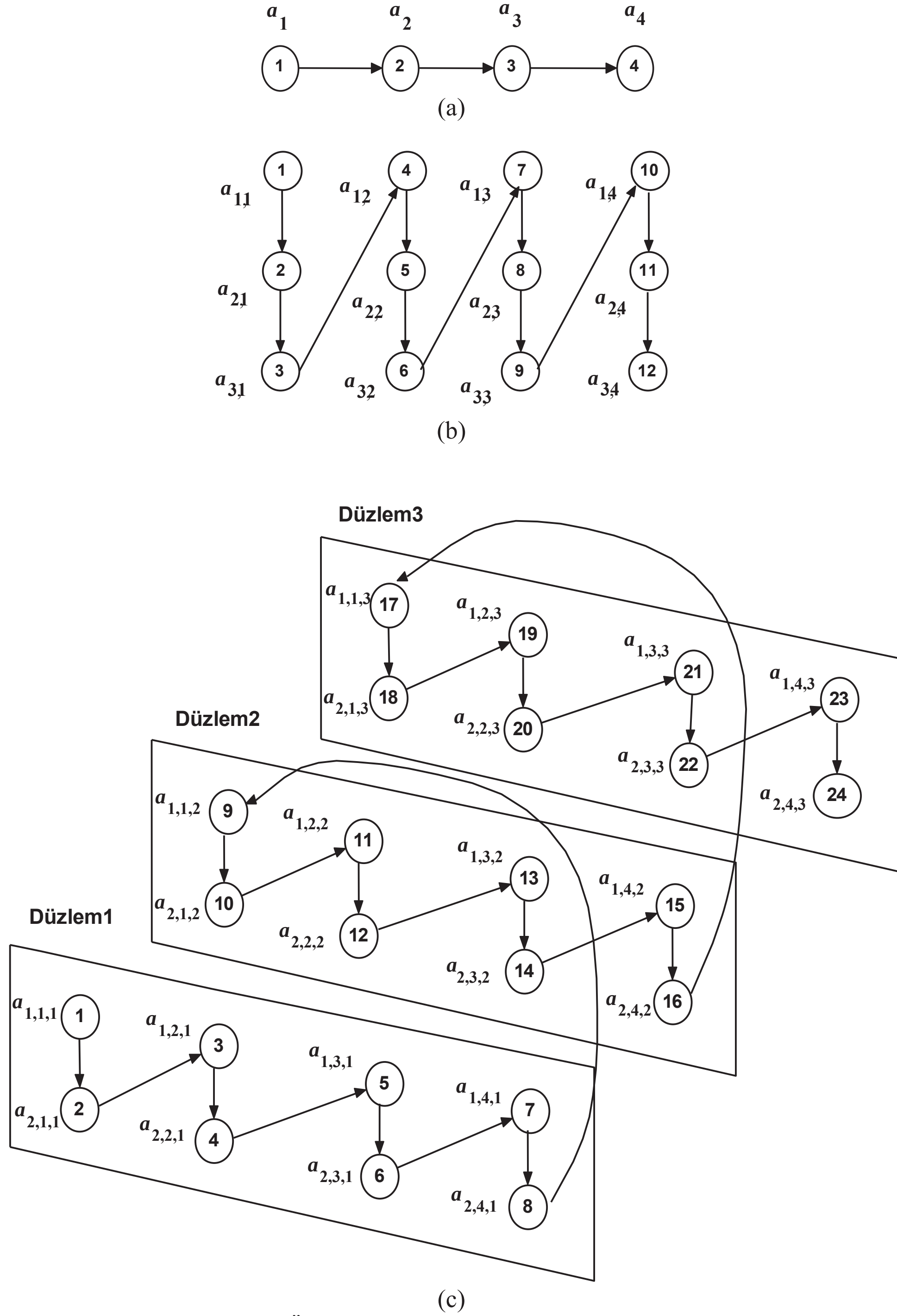
n=**i**+(**j**-1)*L1+(**k**-1)*L1*L2

bağıntısından hesaplanır.

İndisli, özellikle çok indisli, değişkenlerin kullanımında `DIMENSION` ile tanımlanan maksimum boyut aşılmamalıdır. Örneğin `A(100)` olarak tanımlanan bir indisli değişkenin bellek adresi sayısı 100'dür; bu değişenin `A(101)` veya `A(120)` adreslerine veri kaydetmek, okumak veya bunlarla işlem yapmak mümkün olmaz. Çünkü söz konusu bellek adresleri oluşturulmamıştır.

6.2 DO DÖNGÜSÜ

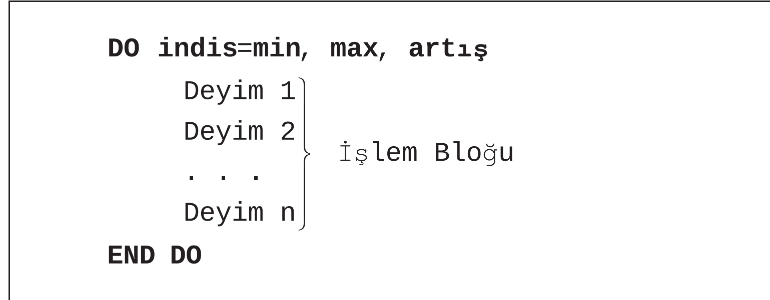
Yapısal programlamanın esas amacı, programın kolaylıkla kodlanabilen ve okunabilen bir yapısının olması ve mantığının kolaylıkla takip edilebilmesidir. `IF` yapısı bu yönde atılan önemli bir aşamadır. Sık kullanılan bir diğer döngü de **DO** döngüsü veya sayaç döngüsüdür.



Şekil 6.3 a) Tek, b) Çift, c) Üç indisli değişkenin bellekte depolanma sırası. ifadesiyle bulunabilir.

6.2.1 DO DÖNGÜSÜNÜN YAPISI

DO döngüsünün yapısı, girdi deyimi **DO** olan ve **END DO** ile kapatılan blok deyim gurubudur. DO döngüsünün genel kullanım şekli



```
DO indis=min, max, artış
     Deyim 1 ⎫
     Deyim 2 ⎪
     . . .   ⎬ İşlem Bloğu
     Deyim n ⎭
END DO
```

olarak verilir. Burada **indis** döngüde kullanılan sayaç indisidir; sayaç değeri **min**'den başlar ve **max** değerine kadar **artış**'lık artırımlarla (yani indisin artırım miktarı) değişir.



Programın okunabilirliği ve takip edilebilirliğini kolaylaştırmak amacıyla DO *döngüsü içinde yer alan işlem bloğunu 2 veya daha fazla karakter boşluk bırakarak içeriden yazınız.*

DO döngüsünün özelliklerini aşağıdaki şekilde sıralayabiliriz:

1. Döngü parametrelerinin her üçüde (**min, max, artış**) sabit, değişken veya matematiksel bir ifade olabilir. Değişken veya ifade olması halinde, ifadelerin işlemleri önce yapılır; sonra döngü bloğu işlemlerine geçilir.

```
m=100
DO i=1,m,2    ! i=1, 3, 5, . . ., 97, 99
  ...         ! değerlerini alır
END DO


DO deger=1,2*m-1
  ...                ! deger=1,2,3,. . .,199
END DO               ! değerlerini alır


INTEGER :: a, b, c, d
...
DO j=MIN(a,b),MAX(c,d),MIN(a,b,c,d)
  ...
END DO
```

Bu örnekte Fortran arşive fonksiyonları MIN ve MAX kullanılmıştır.

```
DO j=3,-3,-2            ! j=3, 1, -1, -3
  ...                   ! değerlerini alır
END DO
```



Döngü indisi (**indis**) *ve döngü parametreleri* (**min**, **max** *ve* **artış**)*TAMSAYI olmalıdır.*

2. DO döngüsü başında **indis=min** olarak alınır. Eğer **indis*artış≤max*indis** ise program işlem bloğundaki deyimleri icra eder.
3. DO döngüsünün *parametreleri* döngü içinde *değiştirilmemelidir*. Bu kurala döngü değişkeni (**indis**), başlangıç (**min**), son (**max**) ve artırım (**artış**) değerleri dahildir. Örneğin, aşağıdaki programda döngünün içinde **n**, **ir** ve **nmax**'ın değerlerini değiştirme girişimi bu kuralı *ihlal* etmektedir.

```
...
DO n=1,nmax,ir
     ...
     nmax=nmax+3    ! üst sınırı değiştirme girişimi
     ir=ir+1        ! artırım miktarını değiştirme girişimi
     n=n+1          ! döngü indisini değiştirme girişimi
     ir=3           ! artırım miktarını değiştirme girişimi
...
END DO
```

4. Her DO deyimine karşılık bir END DO kapatma deyimi eşlik etmelidir!

```
DO i=1,10
 ...
   DO j=1,25
      ...
   END DO
 ...
END DO
```

5. Bir sonlu toplam hesaplanırken ($\sum_{n=1}^{50} X_n$) indisli değişkenlerden faydalanarak ve DO döngüsü kullanarak, basitçe

```
REAL :: toplam
REAL, DIMENSION(50) :: X
toplam=0.0
DO i=1,50
     toplam=toplam + X(i)
END DO
```

şeklinde yapılabilir.

6. DO döngüsünün içine sadece DO deyimi ile girilir. Asla GO TO deyimi ile döngü içine gönderme (yönlendirme) yapılamaz. Aşağıdaki bu kuralı ihlal eden bir program verilmiştir. (*Fortran 2003'de* GO TO *deyimi tamamen çıkarılmıştır.*)

```
GO TO 600
...
DO  M=l,19
     ...
     600  T=T+X
     ...
END DO
```

7. Herhangi bir durumda DO döngüsünden çıkmak **EXIT** deyimi ile mümkündür. Örneğin, aşağıdaki programda **i>=10** koşulu sağlandığında, döngüden çıkış yapılır ve döngü sonunda i=10 değerine sahiptir.

```
INTEGER :: i
DO i=1,16
   ...
  IF(i>=10) EXIT
   ...
END DO
WRITE(*,*) i
```

8. İşlem bloğunun sonunda END DO'ya ulaşıldığında **indis**'in bir sonraki değeri **indis=indis+artış** ifadesinden hesaplanır. Eğer **indis*artış** halen **max*indis** değerinden küçükse, işlem bloğu icra edilir; değilse döngü işlemleri son bulur.

9. Döngü indisinde **artış** tanımlanmaz ise, otomatik olarak **artış=1** olarak alınır.

10. Döngü bloğu, END DO deyimi ile kapatılır ve döngünün çevirim sayısı, **SAYI=(max-min+artış)/artış** ifadesinden hesaplanabilir. Örneğin,

```
DO j=5,17,2
 . . .
END DO
```

döngüsünde DO-END DO içindeki blokta yer alan ifadelerdeki J değeri 5'ten başlayarak 17'ye kadar 2'şer artırılır; yani j 5, 7, 9, 11, 13, 15, 17 değerlerini alır. Çevirim sayısı ise (17-5+2)/2=7 olmaktadır.

```
DO i=5,4
  ...
END DO
```

döngüsünde (5-4+1)1=0 olacağından, döngü işlem bloğu atlanacaktır.

6.2.2 İNDİS-ÜÇLÜLERİ VE İNDİSLİ DEĞİŞKEN ALT KÜMESİ TANIMLAMA

Şu ana kadar hesaplamalarda indisli değişkenlerin ve tüm indislerin kullanımına değindi. Bir indisli değişkenin alt gurupları veya alt kümesini tanımlamak, bunlarla hesaplar yapmak da mümkündür. Bu işlem *indis-üçlüleri* ile olanaklıdır. Genel kullanım şekli

```
indis_ilk : indis_son : adım
```

şeklindedir. Burada indisli değişkenin alt kümesinin ilk indisi *indis_ilk*, son indisi *indis_son* ve *adım* indislere uygulanacak artırım adımıdır. Örneğin,

```
INTEGER, DIMENSION(10):: de=(/1,2,3,4,5,6,7,8,9,10/)
```

tanımından hareketle `de(1:10:2)` alt kümesi `de(1)`, `de(3)`, `de(5)`, `de(7)` ve `de(9)` içeren bir indisli değişken oluşturur.

İndislerde kullanılan otomatik işlemler aşağıdaki örnekle açıklanmıştır.

```
INTEGER :: i=3, j=7
INTEGER, DIMENSION(10):: de=(/1,2,3,4,5,6,7,8,9,10/)
de(:)          ! orijinal indisli değişken (1, 2, 3, ...,9, 10)
de(i:j)        ! Otomatik olarak adım=1 olur. (3,4,5,6,7)
de(i:j:i)      ! adım=3 alınarak(3,6) elde edilir
de(i:j:j)      ! adım=7 alınarak sadece (3) elde edilir
de(i:)         ! 3'den sonrakiler adım=1 ile (3,4,5,6,7,8,9,10)
de(:j)         ! 1'den adım=1 ile başlar (1,2,3,4,5,6,7)
de(::i)        ! 1'den 10'a kadar adım=3 alınır. (1,4,7,10)
```

Örneğin,

```
INTEGER, DIMENSION(8)  :: a
INTEGER, DIMENSION(5,4):: b
```

tanımlaması ile oluşturulan indisli değişkenler için, indis üçlüsü kullanımı ile hangi bellek alanlarının kullanıldığı yanda şematik olarak verilmiştir.

`b(1,:)` birinci satırdan oluşan 4 elemanlı indisli değişkeni, **`b(3,:)`** üçüncü satırdan oluşan 4 elemanlı indisli değişkeni temsil etmektedir.

a(3:5)

a(1:5:2)

b(1:3,2:4)

b(:,4)

6.2.3 CYCLE VE EXIT DEYİMLERİ

WHILE ve DO döngülerini kontrol etme amacıyla kullanılan iki komut **CYCLE** ve **EXIT** dir. Bir döngüde CYCLE deyimi kullanıldığında, işlem bloğunda yapılan işlemler durdurulur ve kontrol döngünün başına gider, ancak döngü indisi artırılır ve döngü, indis üst sınırına henüz ulaşmamış ise, kaldığı yerden devam eder. Örneğin,

```
INTEGER :: i
DO i=1,5
     ! i=3 için döngüyü atla
     IF(i==3) CYCLE
     WRITE(*,*)'i=',i
END DO
```

programının çıktısında i=1, i=2, i=4, i=5 değerleri görülür; yani CYCLE komutunun icra edildiği satır ile END DO arasında kalan kısım atlanmış olmaktadır.

Bir döngüde **EXIT** deyimi kullanıldığında, işlem bloğunda yapılan işlemler durdurulur ve döngüden sonraki icra edilebilir ilk deyime geçer. Örneğin,

```
INTEGER :: i
DO i=1,5
     ! i=3 için döngüden çık
     IF(i==3) EXIT
     WRITE(*,*)'i=',i
END DO
WRITE(*,*)'Program sonu! '
```

programının çıktısı aşağıdaki gibi olmaktadır.

```
i= 1
i= 2
Program sonu!
```

Bir şartlı DO döngüsünde CYCLE ve EXIT kullanımını inceleyelim. Aşağıdaki örnekte kullanılan i sayacı $50 \le i \le 59$ olduğunda döngü bloğu atlanmakta, $i > 100$ olduğunda döngüden çıkış sağlanmaktadır.

```
INTEGER :: i=0
DO
     i=i+1
     ! 50≤i≤59 için döngüyü atla
     IF(i>=50.AND.i<=59) CYCLE
     ! i>100 için döngüden çık
     IF(i>100) EXIT
     PRINT*, 'i=',i
END DO
PRINT*, 'Döngü tamamlandı ve i=',i
```

Program çıktısı

```
i=   1
i=   2
  . . .
i=  49
i=  60
  . . .
i= 100
Döngü tamamlandı ve i= 101
```

ÖRNEK 1: N adet sayının aritmetik ortalaması ve standart sapmasını indisli değişken kullanarak hesaplayan bir program yazınız.

```
PROGRAM Ornek1
IMPLICIT NONE
!
!  Değişkenler ve Tanımları
!
!  Ortalama – Girilen sayıların ortalaması
!  SSapma   - Sapmaların kareleri toplamı
!  X        - Ortalaması ve Standard sapması hesaplanacak değerler
!  SS       - Standard Sapma
!
REAL :: N=999
! En fazla 999 adet veri için hazırlanmıştır
REAL, DIMENSION(N):: X
REAL :: Ortalama=0.0,SSapma=0.0,SS
INTEGER :: I
PRINT*,' Sayı adedini giriniz'
READ*, N
DO I=1,N
     PRINT*,' Sayının ',N,'ci değerini giriniz'
     READ*,X(I)
     Ortalama=Ortalama+X(I)
END DO
Ortalama=Ortalama/REAL(N)
DO I=1,N
     SSapma=Sapma+(X(I)-ortalama)**2
END DO
SS=SQRT(SSapma/REAL(N-1))
PRINT *,' Ortalama       = ',Ortalama
PRINT *,' Standard sapma = ',SS
END PROGRAM Ornek1
```

6.2.4 İSİMLİ DO DÖNGÜSÜ

İç içe çok sayıda DO döngüsü kullanıldığında hangi döngünün nerede başlayıp ve nerede bittiğini tespit etmek zorlaşabilir. Bu nedenle Fortran 90/95'de döngülere isimler vermek olanaklı hale getirilmiştir. Döngüye atanan isim döngünün başında ve sonunda kullanılır.

`EXIT` ve `CYCLE` komutları kullanıldığında döngü isminin belirtilmesi gereklidir. Genel kullanım şekli aşağıda verilmiştir:

```
[isim:] DO indis=min, max, artis
              Deyimler
               . . .
              IF(mantıksal_ifade) CYCLE [isim]
                . . .
              IF(mantıksal_ifade) EXIT [isim]
                . . .
          END DO [isim]
```

Örneğin, aşağıda iç içe iki döngü (`DISdongu` ve `ICdongu`) verilmiştir. Döngülerin nasıl isimlendirildiğine dikkat ediniz.

```
DISdongu : DO j=1,20                         ⎫
       . . .                                 ⎪
     ICdongu : DO i=1,15  ⎫                  ⎪
                 . . .    ⎬ İçteki Döngü     ⎬ Dıştaki Döngü
     END DO ICdongu       ⎭                  ⎪
       . . .                                 ⎪
END DO DISdongu                              ⎭
```

İsimli döngülerin içinde `CYCLE` veya `EXIT` komutlarının kullanımını içeren bir örnek de aşağıda verilmiştir:

```
0:  DonguA : DO
1:     DonguB: DO
2:        . . .
3:        IF(a>b) EXIT DonguA      ! Satır 9'a atlar
4:        IF(a==b) CYCLE DonguA    ! Satır 0'a gider
5:        IF(c>d) EXIT DonguB      ! Satır 8'e atlar
6:        IF(c==a) CYCLE DonguB    ! Satır 1'e gider
7:      END DO DonguB
8:   END DO DonguA
```

`EXIT` ve `CYCLE` komutlarının yanına döngü ismi belirtilmezse, çevirim arzulamadığınız bir şekilde gerçekleşebilir.

6.2.5 İÇ İÇE DÖNGÜLERİN YUVALANMASI

Bir `DO` döngüsü tamamıyla bir başka `DO` döngüsü içinde yer alıyorsa, bu şekilde yapılandırmaya *iç içe döngü yuvalama* adı verilir. `DO` döngüleri birbirleriyle çakışmadıkları sürece, sınırsız sayıda döngü yuvalama yapılabilir. Çok sayıda döngü yuvalama yaparken, yuvalama hatasına neden olmamak, kolay izlenebilirlik ve okuma sağlamak için her `DO` döngüsünün işlem bloğunun bir kaç karakter içeriden başlatılması ve/veya isimli döngüler kullanılması tavsiye edilmektedir.

DO döngülerinin doğru yuvalanmalarına ilişkin şematik bazı örnekler aşağıda verilmiştir:

```
DO i=1,8
  ...
  DO j=0,12
      ...
      DO k=5,9
           ...
      END DO
      ...
  END DO
  ...
END DO
```

```
DO i=1,10
  ...
  DO j=0,51
       ...
  END DO
   ...
   ...
  DO k=1,21
       ...
  END DO
  ...
END DO
```

```
DO i=1,20
   DO j=1,15
      DO k=2,9
        ...
        ...
        ...
      END DO
   END DO
END DO
```

DO döngülerinin yuvalanmasına ilişkin kurallar IF-END IF yapılarına uygulanan kurallar ile aynıdır. Örneğin,

```
DO i=1,5
     DO j=1,3
      PRINT*, i,j
END DO
```

döngüsünde içteki döngünün END DO deyimi eksik olduğundan Unmatched Nested DO Construct (Eşleşmeyen DO yuvalanması) şeklinde bir derleme hatası verilir.

ÖRNEK 2: DO döngüsü kullanarak *n!* hesabı yapan basit bir program yazınız.

DO döngüsünün yapısını öğrendikten sonra bu yapının kullanımıyla n! de ardışık çarpma işlemi ile çok kolay bir şekilde hesaplanabilir:

```
PROGRAM Ornek2
IMPLICIT NONE
!  n            : faktoriyeli alınacak değer
!  faktoriyel   : n!
!  i            : indis değişkeni
!
INTEGER :: i, n, faktoriyel
READ*, n
faktoriyel=1
DO i=1, n
     faktoriyel=faktoriyel*i
END DO
PRINT 20, n,faktoriyel
20 FORMAT(5x,i3,'!=',2x,i10,' dir')
END PROGRAM Ornek2
```

Bu programdaki çarpma işlemi sayısı (çevirim sayısı) artık **n** değerine bağlıdır; kıyaslama ve sayaç gerektirmemektedir.

6.3 GİRDİ/ÇIKTI DEYİMLERİNDE DÖNGÜ KULLANIMI

Girdi/Çıktı deyimleriyle beraber, normal işlem bloğu dışında, farklı bir şekilde DO döngüsü kullanımı daha mevcuttur.

6.3.1 İMALI DO DÖNGÜSÜ

Girdi/çıktı deyimleriyle kullanılmak üzere çok özel bir imalı DO döngüsü mevcuttur; indisli değişkenlerin konsoldan veya kütükten okutulmasında, örneğin

```
INTEGER, DIMENSION (99) :: tam
READ(5,*) tam(1),tam(2),...,tam(99)
```

şeklinde 99 kez değişkeni vermek zorunda kalabiliriz. Ancak imalı DO döngüsü ile bu problemin üstesinden gelebiliriz. İmalı-DO döngüsünün genel kullanım şekli, girdi için

```
READ(5,10) (<Değişken listesi>, indis=min,max,artış)
```

ve çıktı almak için

```
WRITE(6,15) (<Değişken listesi>, indis=min,max,artış)
PRINT*, (<Değişken listesi>, indis=min,max,artış)
```

kullanılır. Burada **indis**, **min**, **max**, **artış** aynı işlemsel DO döngüsünde kullanılan anlam ve işlevlere sahiptir. *İmalı döngü parantez içinde yazılmalıdır*. Örneğin,

```
REAL, DIMENSION(4) :: a
WRITE(*,'(1X,4F7.3)')(a(i),i=1,4)
```

deyimi ile

```
WRITE(*,'(1X,4F7.3)') a(l),a(2),a(3),a(4)
```

deyimi işlevsel olarak aynıdır. Örneğin, a(1)=1.12, a(2)=2.24, a(3)=5.68 ve a(4)=9.96 olarak alınırsa, yukarıdaki her iki durumda çıktı

```
         1         2         3
123456789012345678901234567890
   1.120  2.240  5.680  9.960
```

şeklinde hazırlanır. Yazdırılan sayısal değerler bir satıra sığmıyorsa, çıktılar bir alt satıra sarkar.

Ancak **WRITE** deyimini, imalı DO döngüsü yerine aşağıdaki gibi, bir işlemsel DO döngüsü içine yerleştirdiğimizde, her çıktının az veya fazla olmasına bakılmaksızın yeni bir satır açılır.

```
REAL, DIMENSION(4) :: a
DO i=1,4
    WRITE(*,'(1X,4F7.3)')  a(i)
END DO
```

Yukarıdaki örnekte a_1, a_2, a_3, a_4 çıktısı, aşağıda verildiği gibi, yukarıdan aşağı doğru sıralanır:

```
         1
1234567890
   1.120
   2.240
   5.680
   9.960
```

İki indisli değişkenlerdeki uygulamasına ilişkin bir başka örnek aşağıda verilmiştir:

```
REAL, DIMENSION(3,4) :: a
10 FORMAT(4F8.3)
WRITE(6,10) (( a(i,j),j = 1, 4),i = 1, 3)
```

(iç döngü: `a(i,j),j = 1, 4`; dış döngü: `( a(i,j),j = 1, 4),i = 1, 3`)

Bu programın çıktı değerleri aşağıdaki düzende olacaktır.

$a_{i,j}$	j → 1	2	3	4
i = 1	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$
2	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$
3	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$

Örneğin

```
WRITE(*,100) ((i,j,j=1,4),i=1,5)
100 FORMAT(4(1x,i5))
```

programının çıktısı

```
         1         2         3
123456789012345678901234567890
     1     1     1     2     1
     3     1     4     2     1
     2     2     2     3     2
     4     3     1     3     2
     3     3     3     4     4
     1     4     2     4     3
     4     4     5     1     5
     2     5     3     5     4
```

olarak elde edilir.



Girdi/Çıktı amacıyla kullanılan imalı DO *döngüsü FORMAT deyimlerinin hazırlanmasında dikkatli olunmalıdır!*

6.3.2 İNDİSLİ DEĞİŞKENLERE BAŞLANGIÇ DEĞER ATAMA

İndisli değişkenlere genellikle başlangıç değeri atamak durumunda kalırız. Örneğin `abc(10)` indisli değişkeninin tüm adreslerine `3.` değeri atama işlemini

```
REAL, DIMENSION(10):: abc
DO i=1,10
    abc(i)=3.    veya   DO i=1,10; abc(i)=3.; END DO
END DO
```

şeklinde yapılabiliriz. Yukarıdaki değer atama işlemi

```
REAL, DIMENSION(10):: abc
abc=(/3.,3.,3.,3.,3.,3.,3.,3.,3.,3./)
```

veya

```
REAL, DIMENSION(10):: abc=(/3.,3.,3., &
    3.,3.,3.,3.,3.,3.,3./)
```

veya

```
REAL, DIMENSION(10):: abc=(/(3.,i=1,10)/)
```

şeklilerinden herhangi birini kullanarak da yapabiliriz. İndisli değişkenlere başlangıç değeri atamak için iki yöntem mevcuttur:

(1) İndisli değişken boyutu çok büyük değilse, tip tanımlama satırında başlangıç değer atama yapılabilir:

```
REAL, DIMENSION(5) :: vektor=(/5.,2.,4.,3.,-1./)
```

Örneğinde, `vektor(1)=5.`, `vektor(2)=2.`, `vektor(3)=4.`, `vektor(4)=3.`, `vektor(5)=-1.` olarak atanmaktadır.

(2) İndisli değişkene atanacak değer "aynı" ancak çok sayıda ise, her değerin tek tek girilmesi pratik değildir. Ancak bu durumda imalı `DO` döngüsü kullanımı ile başlangıç değeri atanabilmesi olanaklıdır. Değer atanacak indisli değişkenin değerleri "(/", "/)" parantezleri içinde verilir.

```
(/ değer_atama1, ..., indis=min,max,artış /)
```

Örneğin, imalı `DO` döngüsü ile değer atama

```
REAL, DIMENSION(25)::A=(/ (REAL(i),i=1,25) /), &
    B=(/ (1.0,i=1,25) /)
INTEGER, DIMENSION (1000)::indis=(/(i,i=1,1000)/)
```

şeklinde gerçekleştirilebilir. Burada `REAL(i)` komutu ile `i` tamsayısı "gerçek" sayıya dönüştürülmektedir. `i=1,25` ile `A(1)=1.`, `A(2)=2.`, `A(3)=3.`, …, `A(25)=25.` ataması yapılmaktadır. Benzer şekilde `B` indisli değişkenine yapılan atamada `B(1)=1.`, `B(2)=1.`, `B(3)=1.`,…, `B(25)=1.` ataması uygulanmaktadır. Diğer taraftan, `indis` tamsayı değişkenine adresleri sırayla `1`, `2`, …, `1000` değerleri atanır.

`CHARACTER` veya `LOGICAL` veri tipleri için, değişken listesi sabitin tipi ile uyuşmalıdır. Buna rağmen, bir `INTEGER`, `REAL` veya `COMPLEX` değişken listesine aynı tipten bir sabit ile başlangıç değeri atanabilir. Benzer şekilde mantıksal tipteki veriler aynı kurallara bağlı kalmak kaydıyla yapılan atamalara örnekler aşağıda verilmiştir:

```
LOGICAL, DIMENSION(5) :: Lojik=(/ (.FALSE.,i=1,5) /)
COMPLEX, DIMENSION(72):: zz=(/ ((0.,0.),i=1,72)/)
```

İki veya daha fazla indisli değişkene değer atarken, satır vektöründe **RESHAPE** deyimi veya niteliği kullanılır. **RESHAPE** deyiminin genel kullanım şekli

```
Çıktı=RESHAPE(indisli_1, indisli_2)
```

olarak verilmektedir. Burada `indisli_1` şekil verilecek tek boyutlu indisli veriler, `indisli_2` yeni indisli değişkenin boyutudur. Örneğin,

```
matris=RESHAPE( (/1,1,1,1,2,2,2,2,3,3,3,3/), (/4,3/) )
```

ilk tek indisli veriler aşağıdaki gibi 4 satır ve 3 sütundan (4×3) oluşan matris formuna getirilir.

$$matris = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$$

Aynı işlem tanımlama satırında, aşağıdaki şekilde, gerçekleştirilebilir.

```
INTEGER, DIMENSION(4,3) :: matris = RESHAPE( &
     (/1,1,1,1,2,2,2,2,3,3,3,3/), (/4,3/) )
```

Benzer şekilde

```
REAL, DIMENSION(2,2)::a=RESHAPE( (/1,2,3,4/) ,(/2,2/)),&
         b=RESHAPE( (/5,6,7,8/) ,(/2,2/))
```

a ve **b** matrisleri

$$a = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}, \quad b = \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix}$$

olarak tanımlanmaktadır.



Program içinde ve/veya tanımlama kısmında birden fazla boyutlu indisli değişkenleri tanımlamak için **RESHAPE** *deyimi veya niteliğini kullanınız. İndisli değişkenin değerleri bellekte depolanma sırasını takip etmelidir!*

6.4 İNDİSLİ DEĞİŞKENLERİN DİĞER ÖZELLİKLERİ

Bazı koşullar altında, indisli değişkenler skalar değişkenlerle yapılan aritmetik işlemler gibi işleme tabi tutulabilirler. İki indisli değişken *aynı boyuta* sahip ise skalar aritmetik işlemlerde kullanılabilir ve eleman-eleman arasında aritmetik işlem yapılır. Örneğin,

```
PROGRAM indisli_toplam
IMPLICIT NONE
INTEGER :: i
REAL, DIMENSION(4) :: a=(/1., 2., 3., 4. /)
REAL, DIMENSION(4) :: b=(/5., 6., 7., 8. /)
REAL, DIMENSION(4) :: c, d
DO i=1,4
     c(i)=a(i)+b(i)   ! Eleman-eleman toplamı
END DO
d=a+b                 ! İndisli değişken olarak toplamı
WRITE(*,100) 'c',c
WRITE(*,100) 'd',d
100 FORMAT(' ',A,'=',5(F6.1,1x))
END PROGRAM indisli_toplam
```

Programının çıktısı

```
         1         2         3
123456789012345678901234567890
 c=   6.0    8.0   10.0   12.0
 d=   6.0    8.0   10.0   12.0
```

görüldüğü üzere her iki işlem aynı sonucu vermektedir. **d=a+b** deyimi de aynı indise sahip elemanları toplayarak sonucu aynı indisli yeni değişkene atamaktadır. Bu özellikten yararlanarak, bu tarz işlemlerde programı kısa ve anlaşılır tutmak bakımından yararlıdır.

Aşağıdaki program deyimlerini incelediğimizde, indisli değişkenlerin uzunluklarının aynı ancak alt ve üst sınır değerlerinin farklı olduğunu görürüz. Aşağıda toplanan (a ve b) ve toplam (c) indisli değişkenleri aynı indis aralığında yer almadığı için c=a+b işlemi *gerçekleştirilemez!*

```
REAL, DIMENSION(1:4)::a=(/1., 2., 3., 4. /)
REAL, DIMENSION(5:8)::b=(/5., 6., 7., 8. /)
REAL, DIMENSION(91:94)::c
c=a+b           ! İşlemi gerçekleşmez
```

Skalar sabit ile indisli değişken çarpım işlemi kısa notasyonla uygulanabilir.

```
REAL, DIMENSION(4)::a=(/5., 6., 7., 8. /)
REAL, DIMENSION(4)::c
REAL :: b=2
c=b*a
WRITE(*,100) 'c',c
100 FORMAT(' ',A,'=',5(F6.1,1x))
```

işleminin sonucu

```
               1         2         3
      123456789012345678901234567890
       c=  10.0   12.0   14.0   16.0
```

olarak bulunur.

FORTRAN 90/95'de skalar değişkenler için kullanılan kütüphane fonksiyonları (**ABS**, **SIN**, **COS**, **EXP** ve **LOG**) indisli değişkenleri de argüman olarak kabul ederler. İndisli değişkenin her bir elemanına kütüphane fonksiyonu uygulanır. Örneğin,

$$\texttt{a=}\begin{bmatrix}1\\1.1\\1.5\\2.1\end{bmatrix},\quad \texttt{EXP(a)=}\begin{bmatrix}e^{1}\\e^{1.1}\\e^{1.5}\\e^{2.1}\end{bmatrix},\quad \texttt{LOG(a)=}\begin{bmatrix}\ln(1)\\\ln(1.1)\\\ln(1.5)\\\ln(2.1)\end{bmatrix}$$

Örneğin, `x(1)=1.`, `x(2)=2.`, `x(3)=3.`, `x(4)=4.` ve `x(5)=5.` olan indisli değişen için $\sin \mathbf{x}$ 'i hesaplayalım. Aşağıdaki programda, x indisli değişeni skalar bir değişken gibi sinüs argümanında kullanılmıştır:

```
PROGRAM Ornek
IMPLICIT NONE
REAL , DIMENSION(5) :: X = (/1.0,2.0,3.0,4.0,5.0/)
! Elementel fonksiyon
PRINT *,'X in Sinüsü = ',SIN(X)
! Dönüşümsel fonksiyon
PRINT *,' X lerin toplamı = ',SUM(X)
END PROGRAM Ornek
```

Çıktısı

```
 X in Sinüsü =  0.841470957 0.909297407 0.141120002 -0.756802499 -0.958924294
 X lerin toplamı =   15.0000000
```

olarak elde edilir.



İndisli değişkenlere skalar değişkenler gibi aritmetik işlem uygulayabilmek için indisli değişkenlerin boyutları aynı olmalıdır.

6.4.1 WHERE YAPISI

FORTRAN 90'nın geliştirilmesiyle, bu dilde, indisli değişkenlerin bazı durumlarda skalar değişkenler gibi kullanılabileceğine daha önce değinmiştik. Buna örnek olarak, iki indisli bir değişkenin logaritmasını ele alalım:

```
INTEGER :: n=100,m=100
REAL, DIMENSION(n,m)::log_deger, deger
DO i=1,n
   DO j=1,m
     log_deger(i,j)=LOG(deger(i,j))      ⇒ log_deger=LOG(deger)
   END DO
END DO
```

Yukarıdaki her iki örnekte de `deger(100,100)` indisli değişkenindeki sayısal değerlerin doğal logaritmaları alınarak `log_deger(100,100)`'ta saklanmaktadır. İndisli değişkenin tüm değerlerinin değil de, bazı değerlerinin (`deger>0`) logaritmasını almak istediğimizi kabul edelim. Bu durumda, aşağıdaki örnekte olduğu gibi, bir dizi `DO` döngüsü ile beraber `IF` deyimini de kullanmak durumunda kalırız.

```
DO i=1,n
     DO j=1,m
          IF( deger(i,j)>0.0 ) THEN
               log_deger(i,j)=LOG(deger(i,j))
          ELSE
               log_deger(i,j)=-99999.  ! yani −∞
          ENDIF
     END DO
END DO
```

Bu işlemi **WHERE** deyimi ile mantıksal ifade bir kerede kullanarak yapabiliriz. Önce `WHERE` deyiminin FORTRAN 90 dilindeki genel kullanım şekline değinelim:

```
[isim :] WHERE (mantıksal_ifade)
          . . .
          . . .        } Blok 1
        ELSEWHERE
          . . .
          . . .        } Blok 2
        END WHERE [isim]
```

`WHERE` yapısı, `IF` yapısına benzer; mantıksal ifade *doğru* olduğunda blok 1, *yanlış* olduğunda blok 2'deki işlemler gerçekleştirilir. Şimdi yukarıdaki örneği `WHERE` yapısını kullanarak yeniden yazdığımızda, indisleri kullanmaya gerek kalmadan, aşağıdaki şekilde basitlik sağlanır.

```
WHERE (deger>0)
     log_deger=LOG(deger)
ELSEWHERE
     log_deger=-99999.
END WHERE
```

FORTRAN 95'de WHERE yapısı aşağıdaki gibi yeni mantıksal ifadelerin kullanılmasına da olanak sağlar.

```
[isim :] WHERE (mantıksal_ifade_1)
            . . .
                    } Blok 1
            . . .
         ELSEWHERE (mantıksal_ifade_2) [isim]
            . . .
                    } Blok 2
            . . .
         ELSEWHERE [isim]
            . . .
                    } Blok 3
            . . .
         END WHERE [isim]
```

WHERE yapısı sadece koşulun sağlandığı durumda tek satırda kullanılabilir.

```
WHERE (mantıksal_ifade) <Atama deyimi>
```

Bazı örnekler

```
INTEGER, DIMENSION(45):: a_deger, b_deger
WHERE (a_deger<0)
     b_deger = ABS(a_deger)
ELSEWHERE
     b_deger = a_deger*a_deger
END WHERE
```

Yukarıdaki örnekte b_deger indisli değişkeni oluşturulurken, a_deger'in negatif değerlerinin mutlak değerleri alınarak pozitif değerlere dönüştürülüyor; pozitif olan değerlerinin karesi alınarak b_deger'e eşitleniyor.

```
INTEGER, DIMENSION(45):: a_deger, abs_deger
WHERE (a_deger<0) abs_deger=ABS(a_deger)
```

Yukarıdaki örnekte de a_deger indisli değişkeninin negatif değerlerinin mutlak değeri alınıyor ve abs_deger indisli değişkenine eşitleniyor.

6.4.2 FORALL YAPISI

FORTRAN 95 programlama dilinde getirilen bu yenilik ile indisli değişkenlere, eleman bazında, bir dizi işlem uygulamasına olanak sağlar. Üzerinde işlem yapılacak elemanlar, hem *indis* değerleri hem de *mantıksal koşul* ile belirlenebilir. İndisli değişkenin belirtilen indisleri ve mantıksal koşuluna uygun olanlar elemanlara işlemler uygulanır.

FORALL yapısının genel kullanım şekli aşağıda verilmiştir:

```
[isim:] FORALL (in1=üçlü1[,in2=üçlü2,. . .,mantıksal_ifade])
           Deyim_1
           Deyim_2
            . . .
           Deyim_n
        END FORALL [isim]
```

FORALL deyimindeki her indis "alt_indis:üst_indis:adım" üçlüsü ile verilebilir. FORALL yapısına IF ve WHERE yapılarında olduğu gibi isim verilebilir.

Örneğin, FORALL yapısını kullanarak 10×10'luk bir birim matris oluşturalım:

```
REAL, DIMENSION(10,10) :: birim_matris=0.0
. . .
FORALL (i=1,10)
     Birim_matris(i,i) = 1.0
END FORALL
```

Aşağıda verilen bir diğer örnekte 20×20'lik bir matrisin elemanlarının aritmetik olarak tersi alınmaktadır. Ancak elemanlarından bir kaçının sıfır olması ihtimaline karşın, bu işlemi sadece sıfır değerini almayanlara uygulamaktadır.

```
REAL, DIMENSION(20,20) :: abc
FORALL ( i=1:20, j=1:20, abc(i,j)/= 0.)
     abc(i,j)=1.0/abc(i,j)
END FORALL
```

Yani yukarıdaki işlemi aşağıdaki programın dizisinin yerine geçmektedir.

```
DO i=1,20
     DO j=1,20
          IF( abc(i,j)/= 0. ) THEN
               abc(i,j)=1.0/abc(i,j)
          END IF
     END DO
END DO
```

Bu iki uygulama arasındaki fark DO-IF döngülerinin kullanıldığı son örnekte, döngüler sıralı olarak gerçekleştirilmektedir. FORALL yapısında ise rasgele olmaktadır. Örneğin,

```
FORALL (i=2:50, j=1:n-1)
     a(i,j)=SQRT(a(i,j))
     b(i,j)=1.0/a(i,j)
END FORALL

FORALL (i=1:n, j=1:m, i>j) indis(i,j)=ABS(i-j)+4
     ...
```

```
FORALL (i=1:max_d, j=1:max_d, deger(i,j)>0)
     a(i,j)=SQRT(ABS(deger(i,j)))
END FORALL
```

`FORALL` yapısı, `IF` ve `WHERE` yapılarında olduğu gibi, tek satırda da kullanılabilir.

FORALL (*in1=üçlü1[,. . .,mantıksal_ifade]***)** Atama deyimi

6.5 DÖNÜŞÜMSEL ARŞİV FONKSİYONLARI

Dönüşümsel arşiv fonksiyonları bir veya daha fazla indisli girdi ve çıktı değişkeni içeren fonksiyonlardır. İndisli değişkenlere bire-bir uygulanan elementsel fonksiyonların tersine dönüşümsel fonksiyonlar tüm indisli değişkeni kullanır. Çıktı bir indisli değişken ise çıktı indisli değişkeninin boyutu girdilerinki ile aynıdır. Örneğin, **`DOT_PRODUCT`** fonksiyonu iki vektörü (indisli değişkeni) girdi olarak kabul eder ve çıktısı "skalar"dır; diğer bir deyişle, yaptığı matematiksel işlem

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix}, \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}^T, \qquad a \cdot b = a_1 b_1 + a_2 b_2 + a_3 b_3 + \cdots + a_n b_n$$

olmaktadır. FORTRAN 90/95 kütüphanelerinde birçok dönüşümsel fonksiyon mevcuttur. Bunlardan en sık kullanılanlar **Tablo 6.1**'de verilmiştir.

ÖRNEK 3: {1,2,3,4,5}'den oluşan vektörün skalar çarpımını uygun dönüşümsel arşiv fonksiyonu kullanarak hesaplayan bir program yazınız.

Bu iş için en uygun dönüşümsel arşiv programı `DOT_PRODUCT`'dır.

```
PROGRAM Ornek_3
IMPLICIT NONE
REAL , DIMENSION(5) :: X = (/1.0,2.0,3.0,4.0,5.0/)
PRINT *,' X . X skalar çarpımı '
PRINT *,' ',DOT_PRODUCT(X,X)
END PROGRAM Ornek_3
```

Program çıktısı

```
 X . X skalar çarpımı
 55.0000000
```

olarak elde edilir.

Tablo 6.1 Standart FORTRAN derleyicilerinde mevcut dönüşümsel arşiv fonksiyonları.

Fonksiyon	Amaç
ALL(indis)	Mantıksal bir fonksiyondur ve indis indisli değişkeninin *tüm* elemanlarının tüm değerleri .TRUE. olduğunda .TRUE. değerini alır
ANY(indis)	Mantıksal bir fonksiyondur ve indis indisli değikeninin *herhangi bir* elemanının değeri .TRUE. olduğunda .TRUE. değerini alır
COUNT(indis)	İndisli indis değişkeninin .TRUE. değerini alan eleman sayısını verir.
DOT_PRODUCT(A,B)	A ve B aynı boyutlu indisli değişkenler (vektör) olmak üzere bu vektörlere skalar çarpım işlemini uygular ve çarpım sonucunu verir.
MATMUL(A,B)	A ve B iki boyutlu indisli değişkenler (matris) olmak üzere bu matrislere A.B sırasıyla çarpım işlemini uygular ve çarpım matrisini verir.
TRANSPOSE(matris)	matris iki boyutlu indisli değişken (matris) olmak üzere bu matrisin transpozunu verir.
MAXVAL(ind1,ic)	ind1 indisli değişkeninde ic'nin .TRUE. olduğu değerlerden maksimum olan değerini getirir.
MINVAL(ind1,ic)	ind1 indisli değişkeninde ic'nin .TRUE. olduğu değerlerden minimum olan değerini getirir.
SUM(ind1,ic)	ind1 indisli değişkeninde ic'nin .TRUE. olduğu değerlerin toplamını getirir. ic'nin kullanımı isteğe bağlıdır; bu seçenek kullanılmadığı takdirde indisli değişkenin tüm elemanlarını toplar.
PRODUCT(ind1,ic)	ind1 indisli değişkeninde ic'nin .TRUE. olduğu değerlerin çarpımını getirir. ic'nin kullanımı isteğe bağlıdır; bu seçenek kullanılmadığı takdirde indisli değişkenin tüm elemanları bir biri ile çarpılır.

6.6 DÖNGÜ OPTİMİZASYONU

FORTRAN'da bir çevirim oluşturmanın en kolay yolu DO döngüsü ile sağlanır. Programlar çalışma zamanının büyük bir kısmı genellikle döngülerde harcar; bundan dolayı, döngüler optimize edilmelidir. Döngü içinde gereksiz hesaplardan kaçınılmalıdır.

6.6.1 DÖNGÜDEN BAĞIMSIZ İFADELERİN KALDIRILMASI

DO döngüsünün içinde, döngü indisi içermeyen indisli veya indissiz değişkenlerle ile işlemleri gerektiren ifadeler döngü dışında hesaplanmalıdır. Bu teknik ile gereksiz ve tekrarlı hesapların kaçınılmasına neden olunur.

Örneğin,

```
REAL, DIMENSION (28):: b
REAL, PARAMETER :: pi=3.14159
REAL :: toplam
...
DO k=l,28
   toplam = toplam + SIN(pi/12.)*b(k)
END DO
```

şeklinde verilen programda `b(k)` ile çarpılan $\sin(\pi/12)$ terimini 28 kez hesaplamaya gerek yoktur. Bu hesaplama pekala, aşağıdaki örnekte olduğu gibi, döngü dışında yapılabilir ve değeri bellekte saklanabilir ve gerektikçe bu değer kullanılır.

```
REAL, DIMENSION (28):: b
REAL, PARAMETER :: pi=3.14159
REAL :: toplam, sinpi
. . .
sinpi = SIN(pi/12.)
DO k=l,28
   toplam = toplam + sinpi * b(k)
END DO
```

Bir başka örnek olarak da aşağıdaki programı ele alalım:

```
REAL, DIMENSION (115) :: c
REAL :: x=2.359
DO j=1,115
    C(j) = 1.25 * j * x**2 * EXP(-0.5*x*x)
END DO
```

Burada eşitliğin sağ tarafındaki ifade olan $1.25x^2e^{-0.5x^2}$ terimi döngü dışında hesaplanabilir; çünkü bu terim hiç bir şekilde döngü değişkem olan **`j`**'ye bağımlı değildir. Bu basitleştirme aşağıdaki şekilde yapılabilir.

```
REAL, DIMENSION (115) :: c
REAL :: x=2.359, zz
zz = 1.25 * x**2 * EXP(-0.5*x*x)
DO j=1,115
    c(j) = zz * j
END DO
```

İç içe döngülerde döngü değişkeninden bağımsız değişkenli indisli değişkenlerde döngü dışında değerlendirilmelidirler. Örneğin,

```
aa : DO j=1,45
         . . .
      bb:  DO i=1,45
             C(i)=A(i,j) * B(j)
         END DO bb
         . . .
     END DO aa
```

Programında **bb** döngüsünde **j** indisi değişmediği için B(j)'nin değerini elde etmek için 45 kez bellek adresine ulaşılmaktadır. Bu program aşağıdaki şekilde değiştirilebilir:

```
aa : DO j=1,45
          . . .
          gecici = B(j)
     bb: DO i=1,45
             C(i)=A(i,j) * gecici
          END DO bb
          . . .
     END DO aa
```

Yukarıdaki gibi B(j)'nin bellek değeri döngü dışında bulunduktan sonra geçici (gecici) bir değişkene atandığında, bellek değerine bir kez ulaşılarak işlem süresi kısaltılmış olur.

Eğer DO döngüsünün içinde, döngü değeri değişmeyen bir değişkenle test yapılıyorsa, bu durumda test döngü dışına alınabilir. Örneğin,

```
REAL :: C=4.5, B=l.2345
. . .
DO j=1,777
     IF(A>3.14) THEN
          X(j)=C*SIN(B)
     ELSE
          X(j)=C*COS(B)
     ENDIF
END DO
```

programında C ve B'nin değerleri sabittir; yani döngü indisi **j**'e bağlı değildir. Ayrıca döngüde kullanılan kıyaslama değişkeni olan A'nın değeri, döngü içinde hiçbir surette değişmemektedir. Bu durumda IF yapısı ile kıyaslama işlemini döngü dışına aşağıdaki gibi taşımak daha uygun olur. Böylece programda 777 kez, zaman alıcı bir işlem olan, kıyaslama işleminden kaçınılmış olunur.

```
REAL :: C=4.5, B=l.2345, U, V
. . .
U=C*SIN(B)
V=C*COS(B)

IF(A>3.14) THEN
     DO j=l,777
          X(j)=U
     END DO
ELSE
     DO j=l,777
          X(j)=V
     END DO
END IF
```

Bir araya getirme: Eğer bitişik iki DO döngüsü aynı alt ve üst sınır değerlerine sahipse, aynı döngü indisi altında toplanabilir. Bu işlem sadece programı kısaltmakla kalmaz aynı zamanda programın hesaplama süresini de önemli ölçüde azaltır. Aşağıdaki örnekte üç indisli değişkene yapılan atamalar veriliyor.

```
d1 : DO  i=1,800
          A(i)=0.125*REAL(i)
     END DO d1
d2 : DO  j=l,800
          B(j)=1./(1.+REAL(j))
     END DO d2
d3 : DO  k=l,800
          C(k)=SIN(a(k)+b(k))
     END DO d3
```

Bu program tek bir döngü altında, aynı döngü indis değişkeni kullanılarak, aşağıdaki şekilde bir araya getirilebilir:

```
DO  i=1,800
     A(i)=0.125*REAL(i)
     B(i)=1./(1.+REAL(i))
     C(i)=SIN(a(i)+b(i))
END DO
```

ALIŞTIRMALAR

6.1 Aşağıdaki program deyimlerinde varsa hataları tespit ediniz.

```
a)  REAL :: A, B
    DIMENSION A(9), B(1,250)
b)  REAL, DIMENSION (5,5,5,5,5,5,5):: ar1, ar2
c)  M=6
    N=4
    COMPLEX, DIMENSION (N,M):: sayi1, sayi2
d)  INTEGER DIMENSION :: ix(101)
e)  DIMENSION, REAL (26) :: a, b, c
f)  CHARACTER, DIMENSION U(4,5,6,7)
g)  INTEGER, DIMENSION Y(I(3)) :: hall
h)  REAL A, B
    DIMENSION A(3,4,0:3), B(0:2;0:5)
i)  INTEGER :: NU(0:0)
j)  INTEGER :: C(-4: 5)
k)  REAL, DIMENSION (4,3)(l:3) :: a
l)  LOGICAL, DIMENSION (10) :: aa, bb(20)
m)  REAL :: C(3,-5,6), D(4,14)
```

```
n)  INTEGER :: D(N,N)

o)  INTEGER, PARAMETER :: n=-10
    INTEGER, PARAMETER :: m=-1
    REAL, DIMENSION(n:n*m) :: deg

p)  REAL, DIMENSION(20) :: abc=0.

q)  REAL, DIMENSION(10:0,-2) :: Mat1, Mat2

r)  INTEGER, DIMENSION(12,0,-3) :: İŞLEM

o)  INTEGER, PARAMETER :: n=15
    INTEGER, PARAMETER :: m=n**2
    REAL, DIMENSION(n:m) :: matris_A, matris_B
```

6.2 Aşağıdaki DO döngülerinde indis değişkenlerinin alacağı değerlerin listesini yapın.

```
(a)  DO j=l,12          (b) DO NM=5,27,3
(c)  DO k=l,7,2         (d) DO L=3,30,3
(e)  DO m=5,128,5       (f) DO M3=l,78,8
(g)  DO i=1,13,3        (h) DO KKKKKK=1,14,2
```

6.3 Aşağıdaki DO kullanımlarında varsa hatalı olanlarını tespit ediniz.

```
(a)  DO J=1.23             (b)   DO nn=5,5555
(c)  DO K34=1,72           (d)   DO ido=4,5
(e)  DO M=n,2,-1           (f)   DO L=1,1*12
(g)  DO jin=i+n,j+n+3      (h)   DO MEDENI=8,19,14
(ı)  DO K=4,12,2           (i)   DO i=2,3*m,3
(j)  DO JIM=j1,j2,JB
```

6.4 Aşağıdaki DO döngülerinde çevirimin kaç kez uygulanacağını saptayınız.

```
(a)   DO i=6,6               (b)  DO n=5,l
(c)   DO k=l,7               (d)  DO m=l,5,3
(e)   DO a=0.1,3.,0.2        (f)  DO k=15,l,-2
```

6.5 Aşağıda verilen kısa program gurubuna göre formatlı çıktıların sayısal değerleri ile dizilimlerini bulunuz (tip tanımlamaların doğru yapıldığını kabul ediniz).

```
INTEGER, DIMENSION(3,4) :: MM
DO i=1,3
     DO j=l,4
         MM(i,j)=2*i+3*j
     END DO
END DO
```

```
a) WRITE(6,10) (MM(2,j),j=l,4)
   10 FORMAT(1X,10I2)

b) WRITE(6,11)((MM(i,j),j=l,4),i=1,3)
   11 FORMAT(2X,5I3)
```

```
c) WRITE(6,12)(i,i=1,4),(m,(MM(m,n),n=1,4),m=1,4)
   12 FORMAT(2X,4(2X,I2,1X),/,4(2X,I2,1X,4(2X,I3,1X)))
```

```
d) WRITE(6,13)((MM(i,j),j=I,4),i=1,3)
   13 FORMAT(2X,5I4)
```

```
e) DO i=1,14
      WRITE(6,'(5x,5i4)')(MM(i,j),j=i,4)
   END DO
```

```
f) DO i=1,4
      WRITE(6,14) (MM(i,j),j=4,i,-1)
   END DO
   14 FORMAT(10x,5(I3,T10))
```

6.6 Aşağıdaki program parçalarında varsa hataları bulunuz.

```
(a)    INTEGER :: veri(256)
       veri = 0.0
       veri(10:256:10)=100
       WRITE(*,100) veri
       100 FORMAT(1x,10I8)

(b)    REAL, DIMENSION(10,20) :: vek1=2.
       REAL, DIMENSION(10)    :: vek2=5.
       WRITE(*,100) vek1+vek2
       100 FORMAT(1x,10f4.0)

(c)    INTEGER :: i, j
       INTEGER, DIMENSION(10) :: alt1
       INTEGER, DIMENSION(0:9):: alt2
       INTEGER, DIMENSION(100):: girdi= &
             (/((0,i=1,9),j*10,j=1,10) /)
       alt1=girdi(10:100:10)
       alt2=alt1/10
       WRITE(*,100) alt1*alt2
       100 FORMAT(1x,10I8)

(d)    INTEGER, DIMENSION(4) :: liste=(/4,3,5,2/)
       INTEGER, DIMENSION(10):: vek=(/(10*k,k=-4,5)/)
       Vek(liste)=(/1, 2, 3, 4)
       WRITE(*,*) vek

(e)    REAL, DIMENSION (5,5) :: test=0.
       FORALL (i=1:5, j=1:5)
          test(i,j)=1./REAL(i+j-2)
       END FORALL

(f)    INTEGER, DIMENSION(10) :: Bilgi
       Bilgi=(/ 1, 5, -4, 0, 7, -9, 17, -20, -1, 8/)
       WRITE(*,*) Bilgi <= 0.0
```

```
(g)    INTEGER, DIMENSION(10) :: Bilgi
       Bilgi=(/ 1, 5, -4, 0, 7, -9, 17, -20, -1, 8/)
       WHERE (Bilgi>0)
            Bilgi=-Bilgi
       ELSEWHERE
            Bilgi=-2*Bilgi
       END WHERE
       WRITE(*,*) Bilgi
```

6.7 Aşağıda verilen 5×5'lik `matris` aşağıda verilmiştir. Aşağıda verilen indisli değişken kümelerinin şeklini ve içeriğini bulunuz.

$$\texttt{matris=}\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 7 \\ 4 & 5 & 6 & 7 & 8 \\ 5 & 6 & 7 & 8 & 9 \end{bmatrix}$$

(a) `matris(3,:)`, (b) `matris(:2,)`, (c) `matris(1:5:2,:)`,
(d) `matris(:,2:5:2)`, (e) `matris(1:5:2,1:5:2)`

6.8 `GIRDI.DAT` isimli girdi kütüğünün içeriği aşağıda verilmiştir:

```
34   16   22   81   41   33   12
62   87    9   13   56   93   24
27   61   14   66   72   89    2
 5   15   29   80   58   11   93
13   37   26   76   54   86   16
```

```
     INTEGER, DIMENSION(5,7):: mat
     . . .
(a)  DO i=1,5
          READ(15,*) (mat(i,j),j=1,4)
     END DO

(b)  DO i=1,4
          READ(15,*) (mat(i,j),j=1,6)
     END DO

(c)  READ(15,*) ((mat(i,j),j=1,7),i=1,3)

(d)  READ(15,*) ((mat(i,j),j=1,5),i=1,5)

(e)  DO i=1,5
          READ(15,*) mat(i,:)
     END DO

(d)  READ(15,*) mat
```

6.9 Çarpım tablosunu oluşturan bir program yazınızı. (NOT: Çıktıyı uygun format belirteçleri kullanarak aşağıdaki gibi düzenleyiniz.)

```
    1         2         3         4         5         6
123456789012345678901234567890123456789012345678901234567890
     *    1    2    3    4    5    6    7    8    9    10
1     1    2    3    4    5    6    7    8    9    10
2     2    4    6    8   10   12   14   16   18    20
3     3    6    9   12   15   18   21   24   27    30
.    .    .    .    .    .    .    .    .    .     .
  10   10   20   30   40   50   60   70   80   90   100
```

6.10 N elemanlı bir A dizisinde negatif ve pozitif eleman sayısı birbirine eşittir ve bunlar dizide dağınık bir şekilde bulunmaktadır. Yeni bir B dizisinde pozitif sayıları dizinin ilk yarısına, negatif sayıları ikinci yarısına gelişigüzel atayan bir program yazınız.

6.11 Ekrandan temin edilen N sayıyı büyükten küçüğe doğru sıralayan bir program yazınız.

6.12 Ekrandan temin edilen N sayıyı küçükten büyüğe doğru sıralayan bir program yazınız.

6.13 `NxN` boyutlu bir `a` matrisinin (a) köşegeninin üstünde (`I<J`) (b) köşegeninin altında (`I>J`) yer alan elemanlarının (`a(i,j)`) toplamım veren bir program yazınız.

6.14 Fibonacci sayıları $F_n = F_{n-1} + F_{n-2},\ n \geq 2$ için formülü ile hesaplanıyor. İlk iki sayı $F_1 = F_2 = 1$ olduğuna göre, verilen herhangi bir n için ilk n sayıyı ve bunların F_n / F_{n+1} oranını veren bir program yazmak istiyoruz. Bu amaçla bir algoritma hazırlayın, akış şeması çizin ve Fortran dilinde programlayınız.

6.15 A ve B gibi iki tamsayının (A > B) en büyük ortak bölenini (ebob) bulan bir program yazınız. <u>*NOT:*</u> Bu işlemin yapılabilmesi için Öklit Algoritmasını uygulayınız. Öklit algoritmasına göre, en büyük sayı olan A,

$$A = Q_1 \cdot B + R_1$$

şeklinde yazılabilir. Burada Q_1 bölüm ve r_1 ise kalan'dır. Daha sonra benzer şekilde

$$B = Q_2 \cdot R_1 + R_2$$
$$R_1 = Q_3 \cdot R_2 + R_3$$
$$\cdots$$
$$R_{n-1} = Q_{n+1} \cdot R_n + 0$$

olacaktır. En son Q değeri, en büyük ortak bölendir.

6.16 Chebyshev polinomlarının n.ci derece ve x için hesaplayacak bir program yazınız. Aşağıdaki formülleri kullanınız.

$$T_0(x) = 1, \quad T_1(x) = x \quad \text{ve} \quad T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x),$$

6.17 Bir kültürdeki bakteri sayısı başlangıçtaki bakteri sayısının zaman dilimi ile doğru orantılı olduğu bilinmektedir. Matematiksel olarak bakterilerin üreme sayısı

$$P_t = P_0 \left\{ 1 + \frac{at}{1!} + \frac{(at)^2}{2!} + \frac{(at)^3}{3!} + \cdots + \frac{(at)^k}{k!} \right\}$$

ifadesiyle veriliyor. Burada P_t, herhangi bir *t* anındaki bakteri sayısı, P_0, başlangıçtaki bakteri sayısı, *t* zaman (saat) ve *a*, $0 < a < 1$ arasında değişen ve bakteri tipine bağlı bir artış faktörüdür. Artış faktörü 0.065 olduğu bilinen bir bakteri cinsinin 1 ile 24 saat arasındaki çoğalma faktörü (P_t / P_0) yukarıdaki bağıntıyı *k=11*'e kadar kullanarak hesaplayan bir program yazınız.

6.18 TOFAŞ-FIAT Otomotiv gurubunun ürettiği arabaların ikinci el piyasası fiyatları, kullanım yılı, 0 km fiyatı, ve yıllık değer kaybı bilindiği takdirde hesaplanabilmektedir. Uzmanlar arabaların yıllık değer kayıplarının araba modellerine ve yaşlarına göre dağılımını aşağıdaki şekilde vermektedirler:

Model	**1 Yaş**	**2 Yaş**	**3 Yaş**	**4 Yaş üstü**
Şahin S	% 28.6	%7.2	%7.5	%6
Doğan	% 22.0	%9.1	%6.3	%6
Kartal	% 18.2	%8.3	%6.5	%6
Tempra	% 20.4	%14	%9.5	%6
Palio	% 21.3	%13	%11	%6
Albea	% 24.4	%9.2	%7.8	%6
Stilo	% 21.1	%11	%8.2	%6
Marea	% 28.1	%14.	%7.5	%6
Panda	% 29.2	%17	% 8.4	%6

Bu verilere göre herhangi bir model arabanın 0 km fiyatı ve yaşını okuyup, ikinci el fiyatını hesaplayan bir program yazınız. Örneğin, Şahin S'nin 0 km fiyatı 23,000 YTL olsun. Bir yaşındaki Şahin S fiyatı *23000×(1-0.286)=16,422* YTL, iki yaşındakinin fiyatı *23000× (1-0.286)×(1-0.07)=15,272* YTL vs şeklinde hesaplanacaktır.

6.19 *n*.ci dereceden bir polinomun katsayıları $a_1, a_2, a_3, \ldots, a_{n+1}$ olarak verilmektedir. Polinomun derecesinin maksimum 100 olabileceğini göz önüne alarak, katsayıları `KATSAYI.DAT` isimli bir kütükten okuyan ve polinomu

$$p(x) = a_1 x^n + a_2 x^{n-1} + a_3 x^{n-2} + \cdots + a_n x + a_{n+1}$$

ifadesinden ziyade

$$p(x) = ((\ldots((a_1 x + a_2)x + a_3)x + \cdots + a_{n-1})x + a_n)x + a_{n+1}$$

şeklinde programlayan, polinomun değerlerini [-2,2] aralığında 0.1'lik artırımlarla ekrana listeleyen bir program yazınız.

6.20 Bir fabrikada üretilen bilyalardan rasgele 15 örnek seçilerek çapları ölçülmüştür. Bu ölçüm sonuçları (D_i 'ler `i=l(l)15`) sırasıyla mm olarak 12.3, 12.1, 12.2, 12.2, 12.5, 11.9, 12.0, 12.1, 12.1, 12.2, 12.4, 12.9, 12.1, 12.1, 12.2 olarak kaydedilmiştir. Bir program yazarak bilya çaplarının (a) geometrik, (b) aritmetik, (c) harmonik ortalamalarını, (d) standart hata, (e) standart sapma ve varyansını hesaplayınız.

6.21 Bir işletmede üretilen vidalardan rasgele örnekler alınması için iki işçi görevlendiriliyor. Bu işçilerden biri 6 diğeri de 10 örnek alarak gerekli çap ölçümlerini kalite kontrol mühendisine ulaştırıyor. Örneklerin çap ölçümleri (mm cinsinden)

$D1_i$	5.13	5.10	5.22	5.25	5.18	5.19	5.14	5.22	5.21	5.22
$D2_i$	5.14	5.14	5.21	5.18	5.17	5.18				

olarak tesbit edildiğine göre bu iki gurubun (a) birbirinden bağımsız olarak (b) iki gurubun tamamının aritmetik ortalamalarını ve standart sapmalarını hesaplayan bir program yazınız. Not: (b) sıkkındaki hesaplamaları yapmak için

$$\bar{Z} = \frac{n\,\bar{D}1 + m\,\bar{D}2}{n+m} \qquad \mathrm{Var}(Z) = \frac{n\mathrm{Var}(D1) + m\mathrm{Var}(D2)}{n+m}$$

formüllerinden yararlanınız. Burada $\bar{Z}$ örneklerin tümünün aritmetik ortalaması ve Var(Z)'de bütün gurubun varyansını temsil etmektedir.

6.22 Bir dersin öğretmeni dönem içinde yapacağı 3 vize sınavından ilkinin ağırlığının % 10, diğerlerinin % 15, ödevlerin % 20, final sınavının da % 40 olacağını ilan ediyor. Hazırlayacağı bir program ile öğrencilere 100 üzerinden verilen notlar ve ağırlıkları kullanarak geçme notunu saptamak istiyor. Bu nedenle sizin yazacağınız bir programda en fazla 70 öğrenci için öğrencinin numarası `NO` (**`A9`**), adı ve soyadı `ad_ve_soyad` (**`A25`**), vize notları `vize1`, `vize2`, `vize3`, ödev (`odev`) ile final (`final`) verilerini **`F6.1`** formatıyla ve sırasıyla okuyacak, geçme notunu hesapladıktan sonra ekrana sadece numara, ad-soyad ve geçme notu ile sınıfın not ortalamasını notunu) standart sapma ile beraber yazan bir program hazırlayınız.

6.23 Aşağıdaki veri gurubu için (a) aritmetik, (b) harmonik, (c)geometrik ortalamayı hesaplayan bir program yazınız. Veriler programa aşağıdaki şekilde ve formatta kütükten okutulacaktır.

1.23	1.32	1.28	1.66	1.33	1.19
1.27	1.45	1.26	1.53	1.41	1.38

6.24 Aşağıdaki programların çıktılarını bulunuz.

```
(a)  REAL,DIMENSION(3,3):: a, c
     REAL,DIMENSION(3,3):: b=(/1.,3.,-2.,4.,2.,1.,-3.,2.,1./)
     INTEGER :: i, j
     DO i=1,3
        DO j=1,3
           IF(i==j) a(i,j)=1.0
           d(i,j)=b(i,j)+a(i,j)
        END DO
     END DO
     PRINT*, d
 (b) REAL, DIMENSION(6):: a
     REAL, DIMENSION(6):: b=(/6.,2.,-1.,0.,4.,8./)
     REAL    :: topla
     INTEGER :: n=6, i
     DO i=3,n
        topla=topla+b(i)
        A(i-2)=2.*b(i)-1.
```

```
     END DO
     DO i=1,n
        WRITE(6,15) i,a(i),i,b(i)
     END DO
     WRITE(6,*) topla
     15 FORMAT(3x,'A(',i1,')=',F6.3,/, &
               3x,'B(',i1,')=',F6.3)

 (c) INTEGER :: J, K
     DO J=9,45,9
          DO K=2,10,2
               IF(J==K*K) WRITE(6,8) J,K
          END DO
     END DO
     8 FORMAT(9x,3i3)

 (d) REAL, DIMENSION(10) :: tr
     REAL :: a=9., b=5.
     INTEGER :: i, j, is=1
     j=j+3
     tr(j-2)=a
     tr(j-1)=a-b
     DO WHILE (J<10)
        j=j+1
        tr(j)=tr(1)+tr(j-1)
     END WHILE
     DO i=1,10
        PRINT*, 'T(',i,')=',tr(i)
     END DO

(f)  REAL, DIMENSION (8) :: A, B   ! G.DAT'ın İçeriği
     INTEGER :: i
     OPEN(3,FILE='G.DAT')          !    4.
     DO i=1,7                      !    3.
        READ(3,*) A(I)             !   -1.
        IF(I<S) READ(3,*) B(I)     !    0.
     END DO                        !    7.
     A(1)=A(1)+B(2)                !    1.
     DO i=1,7                      !    -2.
        A(i+1)=A(i)+i*B(i)         !    5.
     END DO                        !    6.
     DO i=1,8                      !    -1
        WRITE(*,22) I,A(I),B(I)    !    2.
     END DO                        !    1.
     22 FORMAT(5X,I2,5X,2(F10.4,2X))

(g)  REAL, DIMENSION (3,3)::a
     INTEGER :: n=3, i, j
     REAL    :: x
     OPEN(2,FILE='A.DAT',STATUS='OLD')
     READ(2,*) ((A(i,j),j=1,n),i=1,n)
```

```
    x=0.0                              ! A.DAT'ın İçeriği
    DO i=1,n                           ! 3. 2. -4. 1. 0. -7.
         DO j=1,n                      ! 3. 7. -2. 5. 5.  0.
            IF(A(i,j)<=x) CYCLE
            PRINT*, A(i,j)
            X=A(I,J)
         END DO
        PRINT*, i,j,x
    END DO
```

```
(h)  REAL, DIMENSION (ll,1) :: a
     INTEGER :: i, j, n
     REAL :: X
     OPEN(5,FILE='B1',STATUS='OLD')
     n=ll
     READ(5,*) (a(i,1),i=1,N)      ! Bl'in İçeriği
     DO i=1,n-l                    ! 7. -1. 6. 0.5 2. 1.2
        DO 20 j=i+1,N              ! 14.3 -23. 89. 0.0 2.
           IF(a(i,1)>=a(j,1)) CYCLE
           X=a(j,1)
           a(i,1)=X
        END DO
     END DO
     WRITE(6,15) (a(i, 1), i=1,n)
     15 FORMAT(5X,6(F7.3,1X))
```

```
(ı)  REAL, DIMENSION(0:7) :: b
     REAL :: X, a, b
     OPEN(15,FILE='B2',STATUS='OLD')
     X=a               !           B2'in İçeriği
     READ(15,*) b      !   1.   5.   -3.   4.   0.   2.   -2.
     B(2)=B(1)+X*X
     B(3)=X
     DO i=0,6
        WRITE(*,100) i,b(i+1)
     END DO
     100 FORMAT(2X,'B(',11,')=',F5.0)
```

```
(i) REAL, DIMENSION (0:10) :: X
     X(0)=3.
     X(l)=6.
     DO i=1,5
        X(i+1)=0.5*X(i)+X(i-1)
     END DO
     DO i=0,4
        C=C+(X(i+1)-X(i))**2
        PRINT*, X(i)
     END DO
     PRINT*,
     PRINT*, C
```

```
(j)  REAL, DIMENSION (5,3) :: S
     REAL :: T1, T, OL, OL1
     INTEGER :: M
     OPEN(8,FILE='DEN')
     T1=0                          !   DEN'in İçeriği
     DO K=l,5                      !   30.     55.     67.
        READ(8,*) (S(K,M),M=1,3)   !   81.     72.     30.
     END DO                        !   10.     27.     63.
     DO i=1,5                      !    8.      2.     97.
     T=0.                          !   68.     70.     74.
        DO j=l,3
           T=T+S(i,j)
        END DO
        T1=T1+S(i,j)
     END DO
     OL=0.25*T
     OL1=0.125*T
     PRINT*, OL,OL1

(k)  INTEGER :: n
     REAL, DIMENSION (4) :: a=(/1.,-3.,2.,5./), &
       b=(/3.,4.,1.,-1.1/)
     DO n=l,4
        S=0.0
        S=S+A(N)*B(N)
     END DO
     PRINT*,'S=',S

(l)  REAL, DIMENSION(6) :: vek1, vek2
     vek1=(/ 2., -4., 5., -11., 0., 3. /)
     WRITE(*,*) SUM(vek1)
     WRITE(*,*) PRODUCT(vek1)
     WRITE(*,*) PRODUCT(vek1,MASK=vek1/=0.)
     WRITE(*,*) ANY(vek1>0.)
     WRITE(*,*) ALL(vek1>0.)
     WHERE (vek1>0)
        vek2=2.*vek1
     END WHERE
     WRITE(*,*) SUM(vek2,MASK=vek2>0.0)

(m)  INTEGER, DIMENSION(6,4) :: veri
     FORALL (i=1:6:2, j=1:4, i-j>=0)
          veri(i,j)=i-j+1
     END FORALL
     WRITE(*,20)((veri(i,j),j=1,4),i=1,6)
     20 FORMAT(1x,4I6)
```

6.25 Bir ankete cevap veren kişilere ait bilgiler CEVAP.DAT isimli kütüğe aşağıdaki formatta kaydedilmiştir. (Tamsayılar `INTEGER*1` olarak tanımlanacaktır). Bu verileri kullanarak aşağıdaki şekilde iki istatistiksel tablo oluşturan bir program yazınız.

Sütun	Veri
1-2	Yaşı? Cevapsız 0 (`INTEGER ::` **`yas`**)
3	Cinsiyeti? Kadın=1, Erkek=2 (`INTEGER ::` **`sex`**)
4	Ev Sahibi mi? Evet=1, Hayır=2, Cevapsız=0 (`INTEGER::`**`Ev_sahibi`**)
5-7	Geliri (YTL)? Cevapsız=900, Yok=999 (`REAL ::` **`Gelir`**)
8	Son seçimlerde oy verdiği parti? ANAP=1, AKP=2, CHP=3, DSP=4, DYP=5, Diğer=6, Cevapsız=0 (`INTEGER::`**`parti_tercihi`**)

```
         1         2         3         4         5         6
123456789012345678901234567890123456789012345678901234567890
-------------------------------------------------------------
                          Gelir (YTL)
      ---------------------------------------------------
Yaş       5'ten az    5-9     10-14     15-19    19 üstü
-------------------------------------------------------------
30 altı
30-39
40-49
50-59
60-69
69 üstü
-------------------------------------------------------------
```

```
         1         2         3         4         5         6
123456789012345678901234567890123456789012345678901234567890
-------------------------------------------------------------
         Cinsiyet   Ev Sahibi       Son seçimde oy verdiği parti
         --------  ----------      ---------------------------
Yaş        K   E   Evet Hayır     ANAP  AKP  CHP  DSP  DYP Diğer
-------------------------------------------------------------
30 altı
30-39
40-49
50-59
60-69
69 üstü
-------------------------------------------------------------
```

BÖLÜM 7
FONKSİYON/ALT PROGRAM OLUŞTURMA VE KULLANMA

Programlar daha karmaşık ve detaylı hale geldikçe programlamada bazı problemler ortaya çıkmaya başlar:

- Karmaşık problemleri çözmek için gerekli algoritmaları tasarlamak, yapısal bütünlük oluşturmak, tüm alternatif ve/veya ayrıntıları hesaba katmak daha da zorlaşır. Özellikle programı kullanacak kişi(ler) programı yazan kişi değilse, kullanıcının programa hatalı veri girişi, hatalı seçenekler veya yanlış seçenek kombinasyonu kullanması vb hallerde kullanıcıya gerekli uyarıları da açık ve net bir şekilde bildirmeyi program tasarımında göz önüne almak gerekir;
- Çoğu kez program algoritması çok iyi tasarlanmış, net bir şekilde bilinse bile, yazılacak programın uzun olması durumunda, programlama dilinde kodlamak zaman alır ve zorlaşır;
- Programlar uzayıp daha karmaşık hale geldikçe, programda yapılan mantık hatalarını bulmak da güçleşir (çok tecrübeli programcıların bile, uzun ve hacimli programları yazarken birçok sentaks ve çalıştırma hatası yapılması olağandır; bu nedenle, hataları bulmak da git gide güçleşir);
- Özellikle başkalarının kullanacağı programlar için, programda sağlanan girdi ve çıktı seçenekleri, bu seçeneklere göre neyin nasıl hesaplandığının anlatıldığı dokümantasyonların hazırlanması gerekir. Büyük ve uzun programlarda gerekli dokümantasyon hazırlama çabaları da artar ve karmaşıklaşır.

Bu problemler, sadece belirli işleri görmek için yazılan *alt programlar* (**`FUNCTION-SUBROUTINE`**) kullanarak kısmen azaltılabilir veya tamamen ortadan kaldırılabilir.

Bir alt program kendi başına *bağımsız* bir programdır; ana program ve/veya başka bir alt program tarafından kullanılabilir (*çalıştırılabilir* veya *çağırılabilir*). Bir alt program girdileri, yani yapacağı işlemlerde kullandığı değerlere alt programın *argümanları* denir, ve değerlerini alt programın çağırıldığı ana veya alt programdan alır, hesap veya işleme tabi tutar ve çıktıları (sonuçları) onu çağıran programa `END FUNCTION/END SUBROUTINE` deyimiyle geri gönderir.

FORTRAN programlama dilinde kullanılan **`ABS`**, **`SQRT`**, **`EXP`** v.b. fonksiyonlar aslında birer **`FUNCTION`** alt programlarıdır. Bu alt programlara, *esas* veya *temel* (*intrinsic)* fonksiyonlar da denir. Bu tür alt programları programcının hazırlamasına gerek yoktur; FORTRAN derleyicileri, standart alt program kütüphanesi veya arşivi içermektedir ve bu arşivde söz konusu fonksiyonlar ve ileride değineceğimiz alt programlar mevcuttur. Programcının hazırladığı, standart arşiv alt programlarının dışındaki, alt programlara *kullanıcı-tarafından tanımlanmış* (*user-defined*) alt programlar denir. Bir programcı belirli işleri gören alt programları çok sık kullanıyorsa, hazırladığı bu alt programlar ile kendi arşivini oluşturabilir.

Alt programlar FUNCTION ve SUBROUTINE olarak iki kategoriye ayrılırlar. Bu programların arasındaki benzerlik ve farklılıklara ileride ayrıntılı olarak değinilecektir. Ancak en önemli farklılık, kısacası, FUNCTION'lar çağırılan programlarda, **ABS**, **SQRT**, **EXP** v.b. arşiv fonksiyonları gibi aritmetik işlem bloğunda kullanılır ve genellikle bir *tek sonuç* üretir. SUBROUTINE'ler çağıran programda **CALL** deyimi ile çağırılır veya devreye sokulur; ayrıca çağıran programa *birden fazla sonuç* gönderebilir.

FUNCTION ve SUBROUTINE kullanmanın avantajları;

- Her alt program, ana programdan bağımsız olarak, hazırlanabilir, derlenebilir ve test edilebilir. Böylece bir işlemi yapmak üzere hazırlanan bir program parçasının, büyük hacimli bir programa yerleştirilmeden önce doğru çalışıp çalışmadığı teminat altına alınır.

- Alt programlar, işlev ve amacına uygun başka programlarda da kullanılabilir; örneğin, bir programcı bir dizi isimi alfabetik sıraya koyma işlemi için hazırladığı bir programı (bir matrisin tersini alan veya iki matrisin çarpımını gerçekleştiren alt program vb) bu işleme ihtiyaç duyduğu programlarda kullanabilmesine olanak sağlar. Bu şekilde *çok amaçlı* olarak "*kullanılabilir*" programların toplam programlama çabalarını azaltır ve program hatalarının ayıklanmasını basitleştirir.

7.1 SUBROUTINE ALT PROGRAMLARI

Bir alt program türü olan SUBROUTINE'lerin kullanım amacı açısından farkı, bu tür alt programlar birden fazla çıktı değeri hesaplamak için kullanılabilirler. Ama diğer yönlerden (ana programdan çağırma şekli hariç) FUNCTION alt programlardan pek farkı yoktur. Bir SUBROUTINE çağıran programda CALL deyimi ile devreye sokulur. Aynı FUNCTION alt programlarda olduğu gibi, SUBROUTINE'ler bir tanımlama deyimi ve bir de END SUBROUTINE gibi sonuçları ana programa gönderen deyim içerir. SUBROUTINE oluşturma genel olarak

```
SUBROUTINE alt_program_ismi(arg_1, arg_2,...,arg_n)
! Giriş/Çıkış Argümanları ile Değişkenlerin Tip tanımları
. . .
! İşlem kısmı
. . .
END SUBROUTINE alt_program_ismi
```

deyim gurubu ile verilir. Buradaki alt_program_ismi ana programda bir değişken ismi olarak atanmamalıdır. Dolayısıyla, alt program ismi için tip tanımlamasına da gerek yoktur; fakat bu isim 31 karakteri *aşamaz*. Alt program isminin ilk karakteri alfabetik bir harf olmak üzere, harf ve rakamlardan oluşabilir. Argüman listesi skalar ve/veya indisli değişkenlerden oluşabilir. Program içinde ana program dışında ayrı bir yere, genelde ana program sonuna (yani END PROGRAM deyiminden sonra) veya MODULE içine yerleştirilir.

SUBROUTINE alt programları, *tam* ve diğer programlardan *bağımsız* programlar olduğundan, bir programın tüm ana özelliklerini içermelidir (Program başı/sonu, değişkenlerin tanımlanması, işlem bloğu vs). Aşağıda bir SUBROUTINE örneği verilmektedir:

```
SUBROUTINE Buyuk(A,B,C,En_Buyuk,Toplam) }   Program Başı
IMPLICIT NONE
!  Girdiler : A, B, C
!  Çıktılar : En_Buyuk, Toplam
REAL, INTENT(IN) :: A,  B,  C
REAL, INTENT(OUT):: En_Buyuk, Toplam         Tanımlama Bloğu
REAL  :: ABC
ABC=A+B+C
En_Buyuk=A
IF(B>En_Buyuk) En_Buyuk=B                    İşlem Bloğu
IF(C>En_Buyuk) En_Buyuk=C
Toplam = A+B+C
END SUBROUTINE Buyuk                      }   Program Sonu
```

Bu örnekte, A, B ve C gibi üç gerçek sayının en büyüğünü bulmak, ve bu sayıların toplamını hesaplamak amaçlanmıştır. Programın girdileri A, B ve C gerçek sayıları, çıktıları da En_Buyuk sayı ve Toplam'dır. Programın argüman listesindeki değişkenlerin hangilerinin girdi (**IN**put) hangilerinin çıktı (**OUT**put) olduğu, tip tanımlama bloğunda INTENT(IN) ve INTENT(OUT) olarak belirtilmektedir. Argüman listesinde yer almayan, yalnız alt programda kullanılan değişkenler, ABC gibi, programın girdi veya çıktı argümanı olmadığından tip tanımlamada IN veya OUT olarak belirtilmez. Bu değişkenlere *yerel değişkenler* adı verilir. Son olarak END SUBROUTINE deyimi ile ana veya çağıran programa, çağırıldığı satırdan bir sonraki satıra geri döner.

Bu örnekteki alt programı kullanan bir ana program şu şekilde verilebilir:

```
PROGRAM alt_program_ornegi
IMPLICIT NONE
REAL :: Sayi_1, Sayi_2, Sayi_3, Maximumu, Toplami
PRINT*, 'Üç adet sayı giriniz'
READ*, Sayi_1, Sayi_2, Sayi_3
!  Alt programı çağır
CALL Buyuk(Sayi_1, Sayi_2, Sayi_3, Maximumu, Toplami)
! Çıktıları yazdır
PRINT*, 'En büyük sayı = ', Maximumu
PRINT*, 'Toplamı       = ', Toplami
END PROGRAM alt_program_ornegi
```

7.1.1 INTENT NİTELİĞİNİN KULLANIMI

Bir SUBROUTINE'nin değişken listesinin tanımlanmasında, *niyet* veya *amaç* anlamına gelen INTENT niteliği, bu değişkenlerin *girdi*, *çıktı* yoksa *hem girdi hem de çıktı* amacıyla mı kullanıldığını belirtmek için kullanılır. INTENT niteliğinin amacı parantez içinde belirtilir ve aşağıda verilen üç durum söz konusu olabilir:

INTENT(IN) Girdi içeren sabit/değişkeni alt programa aktarmak için kullanılır.

INTENT(OUT) Çağıran programa çıktı değerlerini aktarmak için kullanılır.

INTENT(INOUT) Alt programa hem girdi aktaran hem de çağıran programa çıktı göndermek amacıyla kullanılan değişkenlere atanır

INTENT(IN) olarak tanımlanmış değişkenlerin değerlerini alt program içinde değiştirmek mümkün değildir. Ancak INTENT(INOUT)olarak tanımlanmış değişkenlerin değerlerini alt program içinde değiştirilebilir.



Bir SUBROUTINE *alt programının argüman listesindeki değişkenleri mutlaka* **INTENT** *niteliğini belirtiniz!*

INTENT(INOUT) kullanımını ele alan bir örnek aşağıda verilmiştir. Alt programa gönderilen x ve y değişkenlerinin sayısal değerlerinin kendi aralarında değiştirilmesi işlemini yapmaktadır. Bu işlemi yapmak için şöyle bir algoritma uygulayacağız: x=3 ve y=5 olsun; y=x dersek y=3 olur. Daha sonra x=y dersek, y'nin değeri 3 olduğu için x'in değeri de 3 olacaktır. Değişkenlerin sayısal değerlerin yerlerini değiştirmek mümkün *olmadı*! Bu nedenle yeni bir değişkene (T gibi) daha ihtiyacımız vardır: T=x=3 alırsak, x=y ile x=5 değerini alır. Şimdi y=T ataması ile y=3 değerini alır. Burada x ve y alt programa "girdi" olarak temin edilmektedir; ancak bu değerler girdi değerlerinden farklı değerlere değiştirileceği ve tekrar "çıktı" amacıyla kullanılacağı için, aşağıdaki programda, x ve y hem girdi hem de çıktı amacıyla kullanılmaktadır.

```
SUBROUTINE Degistir(x,y)
IMPLICIT NONE
REAL, INTENT(INOUT) :: x, y
REAL :: T
  T=x     !  T ← x
  x=y     !  x ← y
  y=T     !  y ← T
END SUBROTINE Degistir
```

SUBROUTINE alt programını ana programdan veya bir başka alt programdan çağırma işlemine gelince, SUBROUTINE alt programının adı fonksiyon gibi bir değere atanmadığından, çağırma anlamına gelen **CALL** deyimi kullanılır. Genel kullanım şekli,

```
CALL alt_program_ismi(arg_1, arg_2,...,arg_n)
```

olarak verilmektedir. Bu deyim başka bir program içinde kullanıldığında argümanlar otomatik olarak SUBROUTINE içindeki argümanlarla yer değiştirir. Alt program içindeki işlemler yapıldıktan ve sonuçlar elde edildikten sonra ana programa geri dönülür.

7.1.2 ALT PROGRAMLARA DEĞİŞKEN AKTARIMI

FORTRAN programları, alt programlar ile *referansa-göre-geçiş* adını verdiğimiz bir sistem ile iletişim kurar. Alt programlardaki değişken listesi ile oluşturulan bellek alanları ile alt programa gönderilen değerler bire-bir eşleştirilir. Örneğin, aşağıda verilen programda, ana programdaki `a(i)` indisli değişkeni, alt programdaki `u(i)` indisli değişkeni ile eşleştirilir; yani `a(i)`→`u(i)`. Benzer şekilde ana programdaki `x` ve `sayi` değişkenleri alt programdaki `v` ve `z` değişkenleriyle eşleştirilir (`x`→`v, sayi`→`z`).

```
PROGRAM deneme
IMPLICIT NONE
REAL    :: x, a(5)
INTEGER :: sayi
. . .
CALL alt_program(a,x,sayi)
. . .
END PROGRAM deneme
SUBROUTINE alt_program(u,v,z)
IMPLICIT NONE
REAL, INTENT(OUT)::v
REAL, DIMENSION(4),INTENT(IN)::u
INTEGER, INTENT(INOUT) ::z
. . .
END SUBROUTINE alt_program
```

Hafıza Adresi	Ana Program	Alt Program
001	a(1)	u(1)
002	a(2)	u(2)
003	a(3)	u(3)
004	a(4)	u(4)
005	a(5)	u(5)
006	x	v
007	sayi	z



Ana veya alt programdan, bir alt programa aktarılan argümanların tipleri, indisli değişken ise boyutları bire-bir uyuşmalıdır!

Alt programa aktarılan indisli değişkenlerin boyutu `DIMENSION(5)` gibi açık ve net bir şekilde belirtilmiş ise ana programda tanımlanmış bellekte belirtilen boyut kadar yer ayrılır. Ancak boyut belirtilmez ise, çağıran ana veya alt programdaki boyut "*varsayılan boyut*" kadar yer ayrılır. Örneğin, alt programda `DIMENSION(*)` şeklinde boyut vermek ile `a` indisli değişkeni için 100 gerçek sayı, `b` indisli değişkeni 45 tamsayı bellek alanı ayrılmış olmaktadır.

```
PROGRAM ornek1
IMPLICIT NONE
REAL, DIMENSION(100) :: x_deger
INTEGER, DIMENSION(45) :: n_deger
. . .
CALL alt_program1(x_deger,n_deger)
. . .
END PROGRAM ornek1

SUBROUTINE alt_program1(a,b)
IMPLICIT NONE
REAL, DIMENSION(*) :: a
INTEGER, DIMENSION(*) :: b
. . .
END SUBROUTINE alt_program1
```

FORTRAN derleyicisi ana/alt programlara veri aktarımında kullanılan indisli değişkenlerin alt ve üst sınırlarının uyuşup uyuşmadığını kontrol eder; boyut uyuşmazlığının olması halinde "`array out of bounds`" gibi bir hata mesajı verir.



Alt programlara indisli değişken aktarırken, varsayılan boyut tanımlaması yoluna gitmeyiniz. İndisli değişkenlerin alt ve üst sınırı açık bir şekilde verilmelidir.

`CALL` deyimindeki argümanlara ayrılan sahada aritmetik işlem de yapılabilir. Örneğin,

```
CALL alt_program(2.*x+y,SQRT(x-y),v**2,. . .)
```

burada ilk argüman $2x+y$, ikincisi $\sqrt{x-y}$ ve üçüncüsü v^2 'dir.

ÖRNEK 1: `Mevcut` kişiden oluşan bir sınıftaki öğrencilerin boyları `Boylar(i)` indisli değişkeni ile temsil edilsin. Bu sınıftaki öğrencilerin boy ortalamasını (`Boy_Ort`) ve boyların standart sapmasını (`SSapma`) hesaplayan bir `SUBROUTINE` yazınız. Standart sapma formülü

$$\text{SSapma}=\sqrt{\frac{1}{\text{Mevcut-1}}\sum_{\text{i=1}}^{\text{Mevcut}}\left(\text{Boylar(i)-Boy_Ort}\right)^2}$$

ile verilmektedir.

```
SUBROUTINE Boylar(Mevcut,Boylar,Boy_Ort,SSapma)
IMPLICIT NONE
!              Değişkenler ve Tanımları
!
!  Mevcut       : Sınıf mevcudu
!  Boylar       : Boyların kayıtlı olduğu indisli değişken (Uzunluğu
!                 Mevcut kadardır)
!  Boy_Ort      : Boy ortalaması
!  SSapma       : Standard Sapma
!
INTEGER, INTENT(IN) :: Mevcut
REAL, DIMENSION(Mevcut),INTENT(IN) :: Boylar
REAL, INTENT(OUT) :: Boy_Ort=0.0, Sapma=0.0
INTEGER :: i
!  Ortalama Hesap Bloğu
DO i=1, Mevcut
   Boy_Ort=Boy_Ort + Boylar(i)
END DO
Boy_Ort=Boy_Ort/REAL(Mevcut)
!   Standard Sapma Hesap Bloğu
DO i=1, Mevcut
   SSapma=Sapma +(Boylar(i)-Boy_Ort)**2
END DO
SSapma=SQRT(SSapma/REAL(Mevcut-1))
END SUBROUTINE Boylar
```

ÖRNEK 2: 50 kişilik bir sınıftaki öğrencilerin 1. ve 2. vize (`Vize_1` ve `Vize_2`), ödev (`Odev`) ve final (`Final`) sınavları `NOTLAR.DAT` isimli bir kütükte kayıtlı bulunmaktadır. Bu notları okuyacak, her sınav ve ödev için sınıf ortalamasını ve standart sapmasını hesaplayacak bir program yazınız.

Örnek l'de hazırlanan **BOYLAR** alt programı bu iş için pekala kullanılabilir. Bu durumda, program aşağıdaki şekilde yazılabilir.

```
PROGRAM Not_Ortalama
IMPLICIT NONE
INTEGER :: Mevcut, i
REAL, DIMENSION(50):: Vize_1, Vize_2, Odev, Final
REAL :: Ortalama, Sapma
!   Veri okuma amacıyla kütük açma işlemi
  OPEN(UNIT=10,FILE='NOTLAR.DAT',STATUS='OLD')
!   Verileri oku
READ(10,*) Mevcut
DO i=1,Mevcut
READ(10,*) Vize_1(i), Vize_2(i), Odev(i), Final(i)
END DO
!  1. vize notları ortalaması
CALL Boylar(Mevcut, Vize_1, Ortalama, SSapma)
CALL Yazdır('1.Vize',Ortalama,SSapma)
!  2. vize notları ortalaması
CALL Boylar(Mevcut, Vize_2, Ortalama, SSapma)
CALL Yazdır('2.Vize',Ortalama,SSapma)
!  Ödev notları ortalaması
CALL Boylar(Mevcut, Odev  , Ortalama, SSapma)
CALL Yazdır('Ödev  ',Ortalama,SSapma)
!  Final notları ortalaması
CALL Boylar(Mevcut, Final , Ortalama, SSapma)
CALL Yazdır('Final ',Ortalama,SSapma)
END PROGRAM Not_Ortalama
!
SUBROUTINE Yazdir(CH, Ortalama, SSapma)
IMPLICIT NONE
REAL, INTENT(IN):: Ortalama, SSapma
CHARACTER(LEN=6), INTENT(IN) :: CH
   WRITE(6,5) CH
   WRITE(6,7) Ortalama, SSapma
5 FORMAT(/5X,A6,3X,'   ICIN',/,5X,14('-')/)
7 FORMAT(5X,'ORTALAMA=',F9.6,5X,'STANDARD &
  SAPMA=',F9.6)
END SUBROUTINE Yazdir
```

Bu program ile `Vize_1`, `Vize_2`, `Odev` ve `Final` için 4 kez hesaplanması gereken `Ortalama` ve `SSapma` bir alt program yardımıyla hesaplanmıştır. Her alt program çağrısından çıktıktan sonra çıktı değerleri yazdırılarak, daha az değişken kullanımı ve dolayısıyla daha az *bellek* gerektirmiştir. Benzer şekilde `Yazdir` isimli alt program ile yazdırma işlemi kısaltılmıştır.

ÖRNEK 3: $z = x + iy$ karmaşık sayısının büyüklüğünü $|z| = \sqrt{x^2 + y^2}$ ve argümanını $\arg(z) = \tan^{-1}(y/x)$ bağıntılarına göre hesaplayan bir `SUBROUTINE` hazırlayalım.

Problemi programlamadan önce '*nelere ihtiyacımız var*?' bunları tesbit edelim. Öncelikle, *x* ve *y* değerlerine (girdiler) ihtiyacımız olduğu aşikardır. Bu değerleri kullanarak büyüklüğü `Buyukluk` ve argümanını `Arguman` (çıktılar) hesaplayabiliriz.

```
PROGRAM Sanal_SAYI
IMLICIT NONE
REAL :: x, y, Buyukluk, Arguman
PRINT*, 'X ve Y yi Giriniz '
READ*, x, y
    CALL KOMPLEX(x, y, Buyukluk, Arguman)
PRINT*, 'BÜYÜKLÜK= ',Buyukluk
PRINT*, 'ARGÜMAN = ',Arguman
END PROGRAM Sanal_SAYI
!
SUBROUTINE Komplex(xx, yy, MutlakZ, Arg)
IMPLICIT NONE
REAL, INTENT(IN) :: xx       ! Gerçek kısım
REAL, INTENT(IN) :: yy       ! Sanal kısım
REAL, INTENT(OUT):: MutlakZ  ! Büyüklük
REAL, INTENT(OUT):: Arg      ! Argüman
! Sabit tanımı
REAL :: pi=3.14159
MutlakZ = SQRT(xx*xx+yy*yy)
Arg     = ATAN(yy/xx)   ! Burada ATAN2'de kullanılabilirdi
IF(XX<0.0.AND.YY>0.0) Arg = Arg + pi
IF(XX<0.0.AND.YY<0.0) Arg = Arg - pi
END SUBROUTINE Komplex
```

Oysa problem mesela, karmaşık sayının düzlemde bulunduğu bölgenin tespiti gibi bir ikinci parametrenin hesabını da gerektiren şekilde tasarlamak istersek, bu durumda koordinat sistemindeki bölgeleri `Bolge` (`1`=1.bölge, `2`=2.bölge, v.s) ile temsil ederek, program yeniden aşağıdaki şekilde düzenlenebilir.

```
PROGRAM Sanal_SAYI2
IMPLICIT NONE
REAL :: x, y, Buyukluk, Arguman
INTEGER :: Bolge
PRINT*, 'X ve Y yi Giriniz '
READ*, x, y
    CALL KOMPLEX(x,y,Bolge,Buyukluk,Arguman)
PRINT*, 'BÜYÜKLÜK= ',Buyukluk
PRINT*, 'ARGÜMAN = ',Arguman
PRINT*, ' Sayı ',Bolge, ' ci bölgededir'
END PROGRAM Sanal_SAYI2
```

```
SUBROUTINE Komplex(xx, yy, Bolge, MutlakZ, Arg)
IMPLICIT NONE
REAL, INTENT(IN) :: xx        ! Gerçek kısım
REAL, INTENT(IN) :: yy        ! Sanal kısım
REAL, INTENT(OUT):: MutlakZ ! Büyüklük
REAL, INTENT(OUT):: Arg      ! Argüman
INTEGER, INTENT(OUT)::Bolge ! Bölge belirteci
REAL :: pi=3.14159
MutlakZ = SQRT(xx*xx+yy*yy)
Arg     = ATAN(yy/xx)
IF(XX>0.0.AND.YY>0.0) Bolge=1
IF(XX<0.0.AND.YY>0.0) THEN
    Arg=Arg+pi
    Bolge=2
END IF
IF(XX<0.0.AND.YY<0.0) THEN
    Arg=Arg-pi
    Bolge=3
ENDIF
IF(XX>0.0.AND.YY<0.0) Bolge=4
END SUBROUTINE Komplex
```

7.1.3 SAVE NİTELİĞİ VE DEYİMİNİN KULLANIMI

FORTRAN 90/95 standartlarına göre bir alt programdan `END SUBROUTINE` ile çıkıldığında yerel değişkenler tanımsız hale gelir. Bir program içinde aynı alt program tekrar kullanıldığında, yerel değişkenlerin en son aldıkları değerler aynı olabilir veya olmayabilir. Bu farklılığın nedeni kullanılan derleyicinin özelliklerinden kaynaklanır. Bir alt programdaki "yerel" değişkenlerin her kullanımında, bir önceki çağırılışındaki değerlerini korumak istiyorsak **SAVE** niteliğinden yararlanmalıyız. SAVE niteliği, aynı diğer niteliklerde olduğu gibi, tip tanımlama bloğunda kullanılır.

```
PROGRAM save_ornek
IMPLICIT NONE
REAL :: a, b, c, d
  a=1.0; b=3.0
CALL alt_pr(a,c)
PRINT*, a,b,c
CALL alt_pr(b,d)
PRINT*, b,c,d
END PROGRAM save_ornek
SUBROUTINE alt_pr(a,c)
IMPLICIT NONE
REAL, INTENT(IN) :: a
REAL, INTENT(OUT):: c
REAL, SAVE :: b  ! b'nin yerel değişken olduğuna dikkat edin
b=a*a+b           ! ilk kullanımda b=0'dır
c=a+b
END SUBROUTINE alt_pr
```

Programının `alt_pr` programı ilk kez kullanıldığında `b`→`1*1+0=1`, `c`→`1+1=2` olmaktadır. `b`'nin değeri (yani 1) korunmaktadır. Alt program ikinci kez kullanıldığında `b`→`3*3+1` (önceki kullanımda b=1 idi)`=10`, `c`→`3+10=13` olmaktadır.



Bir alt programda **SAVE** *niteliği ile tanımlanan yerel değişkenler, alt program bir kez daha kullanıldığında, değişkenlerin en son aldıkları değerler korunmuş olur.*

FORTRAN aynı zamanda `SAVE` deyimi şeklinde de kullanılmaktadır. Bir alt programın değişken tanımlama kısmında yer alır ve icra-edilemeyen bir deyimdir. `SAVE` deyimi ile tanımlanan herhangi bir yerel değişkenin değeri, alt programın her çağırılışında değişmez. `SAVE` deyimi ile değişken tanımlanmamış ise tüm yerel değişkenlerin değeri alt programın çağırıldığında değişmez.

Genel kullanım şekli

```
SAVE :: değişken_1, değişken_2,..., değişken_n
```

veya tanımlama bloğunda kısaca

```
SAVE
```

şeklinde kullanılır.

7.2 FONKSİYON ALTPROGRAMLARI

FORTRAN dilinde müsaade edilen fonksiyon türleri harici, aritmetik deyim ve arşiv fonksiyonları olmak üzere üç türlüdür. Ancak her üç tür için kullanımları neticesinde fonksiyon tek bir sonuç verir.

Örneğin, **ABS**, **SQRT**, **EXP** v.b. arşiv fonksiyonları olduğundan daha önce bahsetmiştik. Program içinde aşağıdaki gibi kullanıldıklarında sonuç tek'tir:

```
Y=SQRT(A*A+B*B)
```

Veya

```
Z=SIN(1.+EXP(-X))
```

ifadeleri `Y` ve `Z` sayısal değerlerine eşittir.

Bu örnekte takip edilen işlem, ilk önce `SQRT` altprogramı çağırılır. Daha sonra `A*A+B*B` değeri hesaplanır ve bu değer alt programın argümanı olarak karekök işleminin tarif edildiği `SQRT` fonksiyonuna aktarır. Alt program çalıştırılır-*yani karekök işlemi hesaplanır*-ve işlem sonucu çağıran programa (ana veya alt program olabilir) geri gönderilir.

Bir *harici* (*external*) fonksiyon `FUNCTION` deyimi ile tanımlanan ve `END FUNCTION` deyimi ile son bulan bir programdır. `FUNCTION` alt programı, herhangi bir arşiv fonksiyonun programda çağırıldığı yöntemle çağırılır ve kullanılır. Çağırma cümlesindeki değişken listesinin yani `FUNCTION` argüman listesindeki değişken listesindeki isimler ile aynı olması gerekmez. Sadece tiplerinin ve sırasının aynı olmasına dikkat edilmelidir. Genel kullanım şekli aşağıda verilmiştir:

```
FUNCTION fonksiyon_ismi (argüman listesi)}    Fonksiyon Başı
IMPLICIT NONE                            }
!  Değişken tip tanımlamaları            }   Tip Tanımlama
. . .                                    }
!  İşlem kısmı                           }   İşlem Bloğu
. . .                                    }
fonksiyon_ismi=ifade                     }   Fonksiyona Değeri atama
END FUNCTION fonksiyon_ismi              }   Fonksiyon Sonu
```

Örneğin,

```
REAL FUNCTION En_Buyuk(A,B,C)
REAL, INTENT(IN) :: A, B, C
  . . .
END FUNCTION En_Buyuk
```

şeklindeki bir `FUNCTION` alt programı, ana programda aşağıdaki gibi kullanılabilir:

```
REAL :: A, T1, T2, T3
. . .
A=En_Buyuk(T1,T2,T3)
. . .
```

bu durumda `A` değişkeni `Tl`, `T2` ve `T3`'ün en büyüğüne eşit olacaktır.

Örneğin, `A`, `B` ve `C` gibi üç adet sayının büyük olanını bulmak için bir fonksiyon programı işlemleriyle beraber aşağıda verilmiştir.

```
FUNCTION En_Buyuk(A,B,C)
IMPLICIT NONE
REAL, INTENT(IN) :: A, B, C
REAL :: En_Buyuk
  En_Buyuk=A
  IF(En_Buyuk<B) En_Buyuk=B
  IF(En_Buyuk<C) En_Buyuk=C
END FUNCTION En_Buyuk
```

Fonksiyon ismi olan `En_Buyuk` gerçek değişkeni alt program çıkışında en büyük değere atanmıştır.

Bir başka örnek de $n!$ hesabı için verilebilir:

```
INTEGER FUNCTION Faktoriyel(n)
IMPLICIT NONE
INTEGER, INTENT(IN) :: n
INTEGER :: i
Faktoriyel=1
DO i=1,n
     Faktoriyel=Faktoriyel*i
END DO
END FUNCTION Faktoriyel
```

`FUNCTION` alt programlarında dikkat edilecek noktalara şunlardır:

- Alt programın ilk satırında mutlaka `FUNCTION` tanımlama cümlesi yer almalıdır.
- Fonksiyon alt programının ismi, değişken isim tanımlama kurallarına tabidir ve bu kurallara uygun olmalıdır; yani en fazla 31 karakter uzunlukta olmalı ve ilk karakter bir harf olmalıdır.
- Bir alt program kendi başına ve bağımsız tam bir programdır. Fonksiyonun argümanları `REAL`, `INTEGER`, `CHARACTER` v.s olarak tipleri tanımlanmalı, normal programlar gibi varsa tip belirteçlerine (`INTEGER`, `REAL`, `DIMENSION` v.b.) sahip olmalıdır.
- Arşiv fonksiyonları ve/veya başka kullanıcı tarafından hazırlanmış alt programlar çağırılıp kullanılabilir.
- Alt programda kullanılan değişken isimleri ve format numaraları *sadece* alt program içinde geçerlidir; bu değişkenler veya numaralar dışarıdaki programlar tarafından algılanmadığından ana ve diğer alt programların değişkenlerinden etkilenmezler.
- Fonksiyonun ana programla ilişkisi sadece fonksiyon için kullanılan argümanlar aracılığı ile olmaktadır. Argüman listesi ile oluşturulan bellek alanları, ana veya diğer alt programlarda, bu listeye atanan değişkenler ile paylaşır.
- Her `FUNCTION` alt programı mutlaka en az bir argümana ve `END FUNCTION` deyimine sahip olmalıdır.
- `FUNCTION`'lar sadece sayısal hesap yapmak için kullanılmazlar; karakter, mantıksal, tamsayı, v.s tiplerini de alabilirler. Bu durumda fonksiyonun tipi `FUNCTION` teriminden önce veya alt program içinde tip bildirme bloğunda belirtilmelidir. Örneğin, uzunluğu 15 karakter olan `A` ve `B` alfa sayısal değişkenlerini argüman olarak alıp, bunları birleştiren bir fonksiyon aşağıdaki şekilde tanımlanabilir:

```
CHARACTER(LEN=31) FUNCTION Birlestir(A,B)
CHARACTER(LEN=15), INTENT(IN) :: A, B
Birlestir=A//' '//B
END FUNCTION Birlestir
```

veya

```
FUNCTION Birlestir(A,B)
CHARACTER(LEN=31) :: Birlestir
CHARACTER(LEN=15), INTENT(IN) :: A, B
Birlestir=A//' '//B
END FUNCTION Birlestir
```

ÖRNEK 4: *C(n,r)* kombinezonunu herhangi bir *n* ve *r* için hesaplayan bir program yazınız.

Bu bölümde, daha önceki kısımlarda, tanımlanmış olan `Faktoriyel` harici fonksiyonunun kullanımıyla ana program oldukça basitleşir. $C(n,r) = n!/r!(n-r)!$

```
PROGRAM kombine
IMPLICIT NONE
INTEGER :: C, N, R, Faktoriyel
PRINT*, 'N ve R yi Giriniz'
READ*, N, R
C=Faktoriyel(N)/(Faktoriyel(R)*Faktoriyel(N-R))
WRITE(*,12) N, R, C
12 FORMAT(3X,'C(',i2,',',i2,')= ',i7)
END PROGRAM kombine
```

Burada `Faktoriyel(N)`$=n!$ `Faktoriyel(R)`$=r!$ ve `Faktoriyel(N-R)`$=(n-r)!$'e karşılık gelmektedir. Daha önce verilmiş olan `Faktoriyel` fonksiyonu kullanılmıştır. Harici fonksiyonlar, aynı `SIN`, `COS`, `EXP` kütüphane fonksiyonları gibi çeşitli aritmetik işlemlere tabi tutulabilmektedirler.

7.2.1 FUNCTION ALT PROGRAMLARINDA İNDİSLİ DEĞİŞKEN KULLANIMI

`FUNCTION` alt programlarında argüman olarak indisli değişken(ler) kullanılabilir. Buna göre, alt programı çağıran cümledeki argümanlarda indisli bir değişken yer almalıdır; yani kullanılan argümanlar ayrıca `DIMENSION` tanımlanmış olmalı, hem alt programda ve hem de ana programda boyutlandırılmalıdır. Örneğin, aşağıda verilen, 25 adet gerçek sayının içinde en büyük değerin bulunması işlemini yapan, programı ele alalım:

```
PROGRAM Buyuk
IMPLICIT NONE
REAL, DIMENSION(25) :: x
REAL  :: Buyuk_Sayi
READ(5,*) x
Buyuk_Sayi=En_Buyuk_Sayi(x)
WRITE(6,20) Buyuk_Sayi
20 FORMAT(1X,'En Büyük Sayı = ',2X,F8.2)
END PROGRAM Buyuk

REAL FUNCTION En_Buyuk_Sayi(a)
IMPLICIT NONE
REAL, DIMENSION(25), INTENT(IN) :: a
INTEGER :: k
En_Buyuk_Sayi=a(1)    ! İşlem bloğu
DO k=2,25
   IF(En_Buyuk_Sayi<a(k)) En_Buyuk_Sayi=a(k)
END DO
END FUNCTION En_Buyuk_Sayi
```

Görüldüğü gibi her iki programdaki indisli değişken boyutu 25'tir. Aslında alt programdaki indisin üst sınırı 25'dir; yani bu indis üst sınırı çağıran programdaki indisli değişkenin maksimum boyutunu aşmamalıdır. Buradan yola çıkarak, alt programlarda değişken boyutlamaya izin verilir. Buna örnek olarak;

```
REAL FUNCTION En_Buyuk_Sayi(a,n)
IMPLICIT NONE
INTEGER, INTENT(IN) :: n
REAL, DIMENSION(n), INTENT(IN) :: a
INTEGER :: k
En_Buyuk_Sayi=a(1)
DO k=2,n
   IF(En_Buyuk_Sayi<a(k))En_Buyuk_Sayi=a(k)
END DO
END FUNCTION En_Buyuk_Sayi
```

programı verilebilir. Bu örnekte görüldüğü gibi `a` indisli değişkeni `DIMENSION(n)` ile `FUNCTION` deyiminde boyutu değişken (çağıran programda belirlenen değerle sınırlı) olarak verilmektedir. Bu durumdaki boyutlamaya *ayarlanabilir boyutlama* adı verilir. Böylece alt programa indisli değişkenin sadece arzu edilen sayıda elemanı aktarılır.

7.3 RECURSIVE ALT PROGRAMLAR

Normalde bir Fortran alt programı içinde, aynı alt program kullanılamaz; diğer bir deyişle, iç içe ardışık bir şekilde kullanılmaz. Ancak bazı problem guruplarında fonksiyonları ardışık olarak kullanmak söz konusu olabilmektedir. Örneğin, faktoriyel ifadesi

$$n! = \begin{cases} n(n-1)! & n \geq 1 \\ 1 & n = 0 \end{cases}$$

şeklinde tanımlanabilir ve bu fonksiyon ardışık bir şekilde programlanabilir. Ardışık işlem yapılması arzulanan alt programın başına, ardışık anlamına gelen **`RECURSIVE`** tip tanımı konur. Aşağıda yukarıda verilen tanıma uygun, faktoriyel hesabını yapan, bir `SUBROUTINE` verilmiştir. Herhangi bir alt programa benzemekle beraber program başında `RECURSIVE` tip tanımına yer verilmiştir ve `Faktoriyel` alt programına, programın kendi içinden de erişilmektedir.

```
RECURSIVE SUBROUTINE Faktoriyel(n, sonuc)
IMPLICIT NONE
INTEGER, INTENT(IN) :: n         ! Faktoriyeli hesaplanacak değer
INTEGER, INTENT(OUT):: sonuc     ! Sonuç → n!
INTEGER :: gecici                ! Geçici değişken
IF(n>=1) THEN
   CALL Faktoriyel(n-1, gecici)
   sonuc = n * gecici
ELSE
   sonuc = 1                     ! n=0 ise
END IF
END SUBROUTINE Faktoriyel
```

Bir fonksiyon kendi kendini çağırıyorsa eşitliğin solunda ve sağında da fonksiyon ismi kullanılacağından bir karmaşaya sebep olacağı kesindir. Fortran dilinde bu karışıklığa yol açmamak için iki farklı isim tanımlamamıza olanak sağlar. Fonksiyonu kullanmak isterken *fonksiyonun ismini*, bir fonksiyon alt programı çıkışında değer atamaya ise *ikinci ismini* kullanırız. Bu özel isim `FUNCTION` deyiminde `RESULT` cümlesi ile belirtilir.

Aşağıda RECURSIVE FUNCTION ile faktoriyel hesabını yapan alt program verilmiştir. RESULT kullanıldığında, fonksiyon ismine (faktoriyel) tip tanımlaması yapılmayabilir; ama ikinci isimin tipi (cevap) tanımlanır.

```
RECURSIVE FUNCTION faktoriyel(n) RESULT(cevap)
IMPLICIT NONE
INTEGER, INTENT(IN) :: n      ! Faktoriyeli hesaplanacak değer
INTEGER :: cevap              ! Sonuç → n!
IF (n >=1) THEN
   cevap = n * faktoriyel(n-1)
ELSE
   cevap = 1                  ! n=0 ise
END IF
END FUNCTION faktoriyel
```

7.4 PURE VE ELEMENTAL ALT PROGRAMLAR

Fortran programlama dili son on on beş yılda paralel işlemcilerde çok daha hızlı kullanabilmek için sürekli değişim göstermiştir. Bu değişimin bir parçası olarak alt programlar, *saf* (PURE) ve *elementsel* (ELEMENTAL) alt programlar adı altında iki türlü sınıflandırmaya tabi tutulmuştur.

7.4.1 PURE ALT PROGRAMLAR

Fortran dilinde kullanılan alt programların argüman listesinde yer alan değişkenler, bazen program içerisinde değiştirilmekte ve giriş değeri bozulabilmektedir. Bazı programlarda arzu edilmeyen bu işlem, programın istenen şekilde çalışmasını etkiler. PURE programların bu tür yan etkileri yoktur; yani programın giriş değişkenlerini, kullanıcının istemi-dışında değiştirmezler. Buna ilave olarak, PURE programların doğası gereği, yerel değişkenlerin SAVE niceliği ile tanımlanmalarına artık gerek yoktur, çünkü PURE olarak tanımlanan program her değişkeni otomatik olarak SAVE niteliği ile korumaktadır. PURE programlar tarafından çağırılan diğer alt programların da PURE olması gerekir.

PURE programların yukarıda bahsedilen yan etkileri olmadığından dolayı aynı derecede icra edilen FORALL yapısını kullanmak güvenlidir. PURE fonksiyon programlarında her argüman INTENT(IN) olarak tanımlanmalı, alt programda STOP deyimi kullanılmamalı ve alt program içinde *giriş/çıkış* (Input/Output) işlemleri yapılmamalıdır. Örneğin, üç sayının aritmetik ortalamasını veren bir PURE fonksiyon aşağıdaki şekilde verilebilir:

```
PURE FUNCTION ortalama(a, b, c)
IMPLICIT NONE
REAL, INTENT(IN) :: a, b, c
REAL :: ortalama
  Ortalama = ( a + b + c )/3.0
END FUNCTION ortalama
```

PURE Subroutine alt programlarının da söz konusu yan etkileri yoktur. Fonksiyon programlarına uygulanan kısıtlamalar bu alt programlar için de geçerlidir, ancak değiştirilmesine müsaade edilebilecek argümanları INTENT(IN) veya INTENT(INOUT) belirteçleri ile tanımlanmalıdır.

Örneğin, üç sayının aritmetik, harmonik ve geometrik ortalamalarını hesaplayan bir alt program aşağıda verilmiştir:

```
PURE SUBROUTINE ortalamalar(a, b, c, &
      Aritmetik, Harmonik, Geometrik)
IMPLICIT NONE
REAL, INTENT(IN) :: a, b, c
REAL, INTENT(OUT)::Aritmetik, Harmonik, Geometrik
  Aritmetik= ( a + b + c )/3.0
  Harmonik = 3.0/( 1.0/a + 1.0/b + 1.0/c )
  Geometrik= ( a * b * c )**(1./3.)
END SUBROUTINE ortalamalar
```

7.4.2 ELEMENTAL ALT PROGRAMLAR

Elementsel alt programlar skalar (tek değerden oluşan) argümanlar için tanımlanır; ancak indisli değişkenler için de kullanılabilir programlardır. Elementsel fonksiyonun değişken listesi skalar ise fonksiyonun sonucu da bir skalar olacaktır. Bu fonksiyonun değişken listesine indisli değişken verilirse, fonksiyonun sonucu da aynı boyutlu bir indisli değişken olacaktır. Örneğin

```
PROGRAM Skalar
IMPLICIT NONE
REAL :: x, Fon
READ*, x
PRINT*, 'x=',x, 'Fonksiyon=',Fon(x)
END PROGRAM Skalar

ELEMENTAL REAL FUNCTION Fon(x)
IMPLICIT NONE
REAL, INTENT(IN) ::x
  Fon=x*x+3.
END FUNCTION Fon
```

Programına `x=1` girildiğinde çıktısı `Fonksiyon=4.000000` olacaktır. Fakat ana programı aşağıdaki şekilde değiştirdiğimizde, her x değeri için fonksiyon işlemlerini yapacaktır.

```
PROGRAM Indisli
IMPLICIT NONE
REAL, DIMENSION(4) :: x=(/1.,2.,3.,4./), Fon
PRINT*, 'x=',x, 'Fonksiyon=',Fon(x)
END PROGRAM Indisli
```

Yani, `x=1` için `4.`, `x=2` için `7.`, `x=3` için `12.`, ve `x=4` için `19.` getirecektir.
`SUBROUTINE` alt programları da `ELEMENTAL` olarak tanımlanabilir.

```
ELEMENTAL SUBROUTINE altprog(a,b,c,d)
```

7.4.3 EXTERNAL DEYİMİ

Bir alt programın ismi, argümanlar listesinde bulunan başka bir alt programı çağırabilmesine olanak veren bir deyimdir. Bunun için kullanılan alt programın ismi çağırılan alt programın argümanlar listesinde yazılması ve Türkçe karşılığı *harici* olan EXTERNAL deyiminde bulunması gerekir. Genel kullanım şekli,

```
EXTERNAL alt_prog_1 [,alt_prog_2,...]
```

olarak verilmektedir. Burada alt_prog bir harici alt program (FUNCTION veya SUBROUTINE) ismidir. Bu deyimin kullanılmasında dikkat edilmesi gereken kurallar şunlardır:

- Bir harici alt program, bir başka alt programın argümanlar listesinde bulunuyorsa, bu alt program ismi ayrıca **EXTERNAL** deyimi ile tanımlanmalıdır. EXTERNAL kullanımına örnek olarak herhangi bir FonkX(x) fonksiyonunun [a,b] aralığında DX artırımlarla değerlerini tablo halinde yazan bir program ele alalım. Program için PROGR isimli bir SUBROUTINE ve FX isimli bir harici fonksiyon kullanılmıştır. Program incelendiğinde alt ve ana programda FonkX 'in, EXTERNAL deyimi ile tanımlandığı görülmektedir.

```
PROGRAM harici
IMPLICIT NONE
EXTERNAL FonkX
REAL :: a=0.0, b=1.0, DX=0.1
   CALL PROGR(FonkX, a, b, DX)
END PROGRAM harici
SUBROUTINE PROGR(FX,T1,T2,T3)
  IMPLICIT NONE
  EXTERNAL FX
  REAL, INTENT(IN) :: T1, T2, T3
  REAL :: X, Y, FX
  INTEGER :: N, i
  N=(T2-T1)/T3+1
  DO i=0,N
   X=I*T3; Y=FX(X); WRITE(6,2) X,Y
  END DO
  2 FORMAT(5X,F7.3,5X,F9.5)
END SUBROUTINE PROGR
REAL FUNCTION FonkX(x)
  IMPLICIT NONE
  REAL, INTENT(IN) :: x
   FonkX=X*X+2.0*X-1.0
END FUNCTION FonkX
```

- Eğer kullanıcının hazırladığı alt program, arşiv fonksiyonlarından herhangi biri ile aynı ismi taşıyorsa, bu alt program isminin EXTERNAL deyiminde verilmesi kullanıcı programının kullanılacağını, arşiv fonksiyonu kullanılmayacağı anlamına gelir.

7.5 MODULE KULLANIMI İLE PROGRAMLAR ARASINDA VERİ PAYLAŞIMI

Bu kısma kadar, ana veya alt programlardan başka alt programlar arasında veri paylaşımının argüman listesindeki değişkenler aracılığı ile yapıldığını öğrendik. Ancak ana veya alt programlara, argüman listesi dışında, değişken girdi/çıktı paylaşımı modül (**MODULE**) oluşturularak da yapılabilir. Bir modül, **MODULE** deyimi ile başlar ve **END MODULE** ile kapatılarak, oluşturulur. Programda her modül için en fazla 31 karakterden oluşan bir isim verilir ve bir programda birden fazla modül kullanılabilir. SAVE deyiminin kullanılması halinde modül içinde yer alan sabit veya değişkenlerin değerlerinin farklı alt programlara giriş/çıkışlarda korunmasını, yani değiştirilmemesini, sağlar. MODULE ayrıca alt programlar gibi derlenebilen bir program birimidir ve programın genellikle başında PROGRAM deyiminden önce yapılandırılır. Başka alt programlarla (FUNCTION veya SUBROUTINE) paylaşılması arzulanan değişken veya sabitlerin tanımlamaları, başlangıç değerleri atama işlemleri, paylaşılan FUNCTION veya SUBROUTINE alt programları yapılandırılabilir. Modüldeki verilerin paylaşılması arzulanan ana veya bir alt programda USE deyimi ile sağlanır. Genel kullanım şekli

```
MODULE isim
  IMPLICIT NONE
  ! ---- Paylaşılması arzulanan sabitler, skalar ve indisli değişkenler
  SAVE
  INTEGER, PARAMETER :: Max_Deger=100, Pi=3.14157
  REAL, DIMENSION(Max_Deger) :: VEKTOR
  REAL :: Toplam, Sapma
  CHARACTER(LEN=12) :: Kutuk_Ismi
END MODULE isim
```

olarak verilir. MODULE içinde tanımlanan sabit ve değişkenlerin değerlerine paylaşılan programlardan erişilebilir.

MODULE'deki sabit ve değişkenlerin değerlerini bir alt programda kullanmak için, alt programda **USE** deyimi ile beraber modülün isminin tanımlanması gerekir.

```
USE modül_1_ismi, modül_2_ismi, . . .
```



Bir programda, farklı alt programlar tarafından **USE** *ile kulanılabilen, birden fazla* **MODULE** *tnımlanabilir.* **MODULE** *içinde tanımlanan sabit/ değişkenlerin kullanıldıkları programlarda tipleri tanımlanmaz.*

Örneğin, bir dairenin çevresini ve alanını hesaplayan bir program ele alalım:

```
MODULE module_kullanimi
   REAL :: pi=3.14159, Alan, Cevre
END MODULE module_kullanimi
!     Modülde pi sabiti ile Alan ve Cevre değişkenleri REAL olarak tanımlandı

PROGRAM Ornek
USE module_kullanimi
!
!     USE  kullanıldığı için aynı sabit ve değişkenleri ana programda
!     tekrar  tanımlamaya gerek yoktur.  R  gibi sadece modülde
!     tanımlanmayan değişkenler burada tanımlanır.
!
REAL :: R
PRINT*, "Yarıçapı girin"
READ*, R
Cevre=2.*pi*R
Alan =pi*R**2
!    Sonuçları ekrana yazdırma işlemi
   CALL Yazdir
END PROGRAM Ornek
!
SUBROUTINE Yazdir
USE module_kullanimi
!
!     Değişkenler bu alt programa modül ile aktarıldığı için hem argüman
!     listesine hem de tip tanımlama deyimlerine gerek kalmamıştır.
!
PRINT*, " ALAN =",Alan
PRINT*, " ÇEVRE=",Cevre
END SUBROUTINE Yazdir
```

Bu programda `MODULE`'de `pi` sabiti ile `Alan` ve `Cevre` değişkenleri tanımlanmıştır. `USE` deyimi kullanılan ana program ile `Yazdir` alt programında kullanıldığı sürece `pi` sabiti ile diğer değişkenlerin son değerleri bu programlara aktarılmaktadır. Dikkat edildiğinde `Yazdir` alt programında argüman listesi verilmemiş, bu alt programa aktarılması istenen değişkenler `USE` deyiminin kullanılması ile aktarılmıştır.

Bir alt programa argüman listesiyle çok sayıda sabit ve/veya değişkenin aktarımı söz konusu olduğunda, temel değişkenler haricinde, aktarılması arzulanan diğer sabit veya değişkenlerin aktarımına uygun bir yapıdır.

7.6 MODULE ALT PROGRAMLAR

Verilerin paylaşılmasının yanı sıra, modüller aracılığı ile `FUNCTION` veya `SUBROUTINE` alt programları da, çeşitli programlar arasında paylaşılabilir; bu alt programlara *modül alt programları* denir. Modül programları, modülün bir parçası olarak derlenir ve `USE` deyimi ile herhangi bir programda kullanılabilir. Modülde tanımlanan alt programlar `CONTAINS` deyimini takiben verilmeli, alt program ismi, değişken listesi ve değişken tanımları verilmelidir. Genel kullanım şekli

```
MODULE modul_1
IMPLICIT NONE
!
!   Paylaşılan verileri bu kısımda tanımlayın

  CONTAINS
    SUBROUTINE alt_program1(a,b,c,x)
    IMPLICIT NONE
    REAL, DIMENSION(10), INTENT(IN) ::a, b
    REAL, INTENT(IN) :: c
    REAL, INTENT(OUT):: x
      . . .
    END SUBROUTINE alt_program1
END MODULE modul_1
```

olarak verilebilir. Herhangi bir programda `USE modul_1` tanımlanırsa `alt_program1` bu programda normal `CALL` deyimi ile kullanılabilir.

Aşağıda bir modül içinde tanımlanan ve paylaşılan bir fonksiyonun kullanımı ele alınmıştır:

```
MODULE modulA
CONTAINS
   REAL FUNCTION fonk(a, b, c)
   IMPLICIT NONE
   REAL, INTENT(IN) :: a, b, c
   fonk = (a*c+b*a+b*c)/(a*a+b*b+c*c)
   END FUNCTION fonk
END MODULE modulA
!
PROGRAM test
USE modulA
IMPLICIT NONE
WRITE (*,*) fonk(3.,1.,2.)
WRITE (*,*) fonk(a=3.,b=1.,c=2.)
WRITE (*,*) fonk(b=1.,c=2.,a=3.)
WRITE (*,*) fonk(3.,c=2.,b=1.)
END PROGRAM test
```

Program çıktısı

```
0.785714269
0.785714269
0.785714269
0.785714269
```

şeklinde elde edilir. Bu örnekte `fonk` fonksiyonunun argüman listesindeki değişkenlerin yerleri atama deyimleri kullanılarak değiştirilmiştir. *Argüman listesinde, atama deyimi kullanıldığında, argüman listesi sırasına uymak zorunluluğu yoktur!*

Bir SUBROUTINE veya FUNCTION alt programını, ana programdan ayrı bir şekilde derleyebileceğimiz ve CALL deyimi ile istediğimiz bir alt programdan çağırarak kullanabileceğimizi biliyoruz. O halde neden bir alt programı MODULE içinde tanımlamak ve USE deyimi ile kullanmak isteyelim? Bu sorunun cevabı özünde şu şekilde açıklanabilir: bir modül ve içindeki alt program derlendiğinde, alt programın arayüzleri ile ilgili tüm ayrıntılar derleyici verilir. Program çağırıldığında derleyici otomatik olarak alt programdaki argüman sayısını, argümanlardan hangilerinin indisli değişken olduğunu ve argümanların INTENT özelliğini kontrol eder. Kısacası, derleyici bir programcının yapabileceği hataların büyük bir kısmını bu esnada yakalar.

Bir modül içinde derlenen ve USE ile erişilen bir alt programın "*bariz arayüz*"ü vardır denir. Alt program her kullanıldığında, derleyici her argümanın tüm ayrıntılarını bilir. Diğer taraftan, modül içinde kullanılmayan alt programların "*kapalı arayüz*"ü vardır denir. Bu alt programlar kullanıldıklarında Fortran derleyicisinin argüman ayrıntıları hakkında hiçbir fikri yoktur; programcının argüman sayısını, tipini ve benzeri bilgileri doğru kullandığını varsayar.

```
MODULE modul_ornek
  IMPLICIT NONE
  REAL :: x = 100.
  REAL :: y = 200.
END MODULE modul_ornek

PROGRAM test
USE modul_ornek
IMPLICIT NONE
INTEGER :: i = 1, j = 2
WRITE (*,'(A25,2I7,2F7.1)') ' Başlama:', i,j,x,y
CALL Alt1(i,j)
WRITE(*,'(A25,2I7,2F7.1)') ' Alt1den sonra:',i,j,x,y
CALL Alt2
WRITE (*,'(A25,2I7,2F7.1)') ' Alt2den sonra:',i,j,x,y
CONTAINS
  SUBROUTINE Alt2
   REAL :: x
   x = 1000.
   y = 2000.
   WRITE (*,'(A25,2F7.1)') ' Alt2 de:', x, y
  END SUBROUTINE Alt2
END PROGRAM test
SUBROUTINE Alt1 (i,j)
IMPLICIT NONE
INTEGER, INTENT(INOUT) :: i, j
INTEGER, DIMENSION(5) :: a
!  Alt2'den önce Alt1'de
WRITE (*,'(2I7)') i, j
CALL Alt2
! Alt2'den sonra  Alt1'de:
WRITE (*,'(2I7)') i, j
a = (/ (1000*i, i=1,5) /)
! Alt2'de indisli değişken tanımından sonra:
```

```
WRITE (*,'(7I7)') i, j, a
CONTAINS
   SUBROUTINE Alt2
   INTEGER :: i
   i = 1000
   j = 2000
   WRITE (*,'(A25,2I7)') 'Alt1deki Alt2de ', i, j
   END SUBROUTINE Alt2
END SUBROUTINE Alt1
```

Programının çıktısı

```
     Başlama:      1      2  100.0  200.0
     1      2
   Alt1deki Alt2de    1000   2000
     1   2000
     1   2000   1000   2000   3000   4000   5000
   Alt1den sonra:      1   2000  100.0  200.0
   Alt2 de: 1000.0 2000.0
   Alt2den sonra:      1   2000  100.0 2000.0
```

şeklinde olur. Çıktının neden bu şekilde elde edildiğini araştırınız.

7.7 INTERFACE (ARAYÜZ) PROGRAGRAMLARI VE ARAYÜZ BLOKLARI

Fortran'ın ileri düzey özelliklerini kullanmak için alt programların "bariz arayüz" olarak kullanılmaları gerektiğinden bahsedildi. Bu şekilde bariz arayüz kullanımı, derleyicinin yakalaması ve düzeltilmesi güç olan hataların belirlenmesinde çok faydalı olmaktadır. Bu arayüzleri oluşturmak kimi zaman çok güçtür; şöyleki 1950'li yıllardan beri bir çok teknik ve mühendislik problemin çözülmesi için yazılan programların büyük bir kısmı Fortran dilindedir. Kısacası Fortran dilinde hazırlanmış belirli problemleri çözmeye yönelik, halen kullandıkları çok sayıda alt program yazılım arşivleri mevcuttur. Bunların yeni Fortran derleyici versiyonlarına adapte etmek oldukça güç ve zaman alıcı bir süreçtir. Fortran 90/95 bu anlamda eski programların harici (*external*) olarak (yazıldıkları versiyonda) derlenmesine ve bunların ayrıntı bilgilerini ana programda arayüz olarak verilmesine olanak sağlar.

Arayüz bloğu harici bir alt programın arayüz karakteristiklerini belirler; böylece Fortran derleyicisi arayüz (interface) bloğundaki bilgileri kullanarak argümanların tutarlılığını kontrol eder.

Bir arayüz bloğu, blok içinde herhangi bir alt programın argümanlarına ait bilgiler tekrar verilerek oluşturulur. Bir arayüzün genel şekli aşağıda verilmiştir:

```
INTERFACE
    Arayüz_1
    Arayüz_2
    . . .
END INTERFACE
```

Her `Arayüz` kullanılan harici alt programın (`FUNCTION` veya `SUBROUTINE`) ilk tanımlama deyimini, argümanlarının tip tanımlamalarını ve `END FUNCTION` veya `END SUBROUTINE` deyimini içerir. Bu deyimler, derleyicinin kullanılan harici alt program ile çağıran programlar arasında tutarlılık kontrollerini yapması için yeterlidir. Örneğin,

```
PROGRAM ARAYUZ_Ornek
IMPLICIT NONE
INTERFACE
    SUBROUTINE Vektor_Siddeti(Boy,a,Siddet) }  İlk deyim
       IMPLICIT NONE                                  }
       REAL, DIMENSION(Boy), INTENT(IN) :: a          }
       INTEGER, INTENT(IN) :: Boy                     }  Tanımlama
       REAL, INTENT(OUT)   :: Siddet                  }
    END SUBROUTINE Vektor_Siddeti   }  Son  Deyim
END INTERFACE
!
REAL, DIMENSION(6):: Vektor=(/ 4.,3.,5.,2.,-3.,9. /)
INTEGER :: Uzunluk = 6
!
CALL Vektor_Siddeti(Uzunluk, Vektor, Siddet)
!
WRITE (*,*) "Siddet=",Siddet
END PROGRAM ARAYUZ_Ornek
```

Arayüz blokları kullanımında dikkat edilecek hususlar:

- Bütün alt programlarınızı `MODULE` içinde veriniz ve **`USE`** deyimi ile kullanınız. Gerekmedikçe arayüz bloğu kullanmaktan mümkün olduğunca kaçınınız.

- `MODULE` içinde tanımlanmış ve `USE` ile kullanıma açılmış bir alt programı, arayüz bloğu içinde tekrar vermeyiniz. (Alt program için iki kez arayüz oluşturmaya çalışır ki derleyici hatası ile sonuçlanır.)

- Arayüz kullanımı, daha ziyade derlenmiş harici (Fortran IV, 77 versiyonları ve/veya C dilinde yazılmış) alt program kütüphaneleri için bariz arayüz oluşturmak amacını taşımaktadır.

- Eski alt programlar için bir arayüz oluşturmanın en kolay yolu, bu programların tümünün bir modül içine yerleştirilmesi ve `USE` deyimi ile çağıran programdan kullanılmasıdır. Örneğin,

```
MODULE Arayuz_Tanimlari
  INTERFACE
     SUBROUTINE Vektor_Siddeti(Boy,a,Siddet)
     IMPLICIT NONE
     REAL, DIMENSION(Boy), INTENT(IN) :: a
     INTEGER, INTENT(IN) :: Boy
     REAL, INTENT(OUT)   :: Siddet
     END SUBROUTINE Vektor_Siddeti
     . . .
     ( Diğer alt program arayüzleri izler  )
     . . .
  END INTERFACE
END MODULE Arayuz_Tanimlari
```

- Her arayüz derleyici tarafından farklı işlenir ve birkaç arayüzde aynı değişken isimlerinin farklı tipte kullanılması çelişki oluşturmaz.

- Arayüzde verilen alt programların argüman listesinde kullanılan değişken isimleri bu alt programların değişken isimleri ile aynı olması zorunluluğu yoktur. Ancak programcının kolaylıkla kafasının karışmasına sebep olabileceği için, farklı isimlendirmelerin yapılması pek tavsiye edilmez.

ÖRNEK 5: İki indisli değişken $a_i = i$ ve $b_i = i^2$ olarak veriliyor. Bu değişkenlerin içeriklerini takas eden bir SUBROUTINE programı yazınız. *Not:* Bu alt programı arayüz olarak tanımlayınız.

Verilerin takas edilmesini içeren bir örnek daha önce verilmişti. Aynı algoritmayı Takas_Et(x,y) alt programında kullanabiliriz. Bizden arayüz kullanmamız istendiğinden, alt programın başlangıç, tanımlama ve son deyimleri INTERFACE-END INTERFACE bloğu içinde verildiğine dikkat ediniz.

```
PROGRAM Ornek5
IMPLICIT NONE
INTERFACE
   ELEMENTAL SUBROUTINE Takas_Et(x,y)  )
   INTEGER , INTENT(INOUT) :: x, y     } Arayüz
   END SUBROUTINE Takas_Et             )
END INTERFACE
INTEGER , DIMENSION(10) :: A, B
INTEGER :: i
DO i=1,10; A(i)=i; B(i)=i*i; END DO
PRINT*, 'Takas işleminden önce ... '
  PRINT*, A; PRINT*, B
  CALL Takas_Et(A,B)
PRINT*, 'Takas işleminden sonra ... '
  PRINT*, A;  PRINT*, B
END PROGRAM Ornek5
```

```
ELEMENTAL SUBROUTINE Takas_Et(x,y)
IMPLICIT NONE
INTEGER , INTENT(INOUT) :: x, y
INTEGER :: Gecici
  Gecici=x
  x=y
  y=Gecici
END SUBROUTINE Takas_Et
```

ÖRNEK 6: Yarıçapı ekrandan girilen on adet dairenin alanını ve çevresini çift hassasiyetli olarak hesaplayan bir program yazınız. Çift hassasiyet ile 15 basamak hassasiyeti ve `pi` sayısını bir `MODULE`'de tanımlayınız; bir arayüz ile beraber kullanılan, alan ve çevre hesaplarını yapan bir alt program oluşturunuz.

Aşağıda oluşturulan modülde çift hassasiyetin 15 basamak olduğu ve pi sayısı verilmiştir. `Cift` tanımlaması ana ve `Alt1` alt programında kullanıldığından her iki programda `USE` deyimi ile modüldeki veriler paylaşılmaktadır. Örnek programın başında `INTERFACE` bloğunda alt program ismi ve tip tanımlamaları verilmektedir. Ayrıca alt programda çevre hesabında kullanılan 2. sabitinin `2.0_cift` olarak verildiğine dikkat ediniz.

```
MODULE Basamak_Hassasiyeti
  IMPLICIT NONE
  INTEGER, PARAMETER:: Cift=SELECTED_REAL_KIND(15,307)
  REAL(Cift), PARAMETER::pi=3.14159265358979_Cift
END MODULE Basamak_Hassasiyeti
!
PROGRAM Ornek6
USE Basamak_Hassasiyeti
IMPLICIT NONE
INTERFACE
     SUBROUTINE Alt1(R,Alan,Cevre)
     USE Basamak_Hassasiyeti
     IMPLICIT NONE
     REAL(Cift),INTENT(IN) :: R
     REAL(Cift),INTENT(OUT):: Alan, Cevre
     END SUBROUTINE Alt1
END INTERFACE
REAL(KIND=Cift):: R, A, C
! REAL(Cift):: R, A, C olarak da tanımlanabilirdi
INTEGER ::i
! Değer oku, hesapla, yazdır
DO i=1,10
     PRINT*,'Yarıçap?'
     READ*,R
     CALL Alt1(R,A,C)
     PRINT *,' Yarıçapı = ',R
     PRINT *,' Alanı    = ',A
     PRINT *,' Çevresi  = ',C
END DO
END PROGRAM Ornek6
```

```
SUBROUTINE Alt1(R,Alan,Cevre)
USE Basamak_Hassasiyeti
IMPLICIT NONE
REAL(Cift),INTENT(IN) :: R
REAL(Cift),INTENT(OUT):: Alan, Cevre
Alan  = pi*R*R
Cevre = 2.0_Cift*pi*R
END SUBROUTINE Alt1
```

ÖRNEK 7: Bir dizi gerçek sayının kareleri ve küpleri toplamını hesaplayan bir program yazınız. N adet sayının kareleri ve küpleri toplamı için bir `SUBROUTINE` alt programı hazırlayınız ve bu alt programı arayüz olarak tanımlayınız

`Istatistik` isimli bir alt program verilmektedir. Bu alt program, sayıların kare ve küplerini en az aritmetik işlem ve en az zaman alacak şekilde, hazırlanmıştır. Alt programın isim ve tip tanımlamalarının `INTERFACE` bloğunun içinde ve ana programda verildiğine dikkat ediniz.

```
PROGRAM Ornek7
IMPLICIT NONE
REAL, DIMENSION (1:5)::A=(/2., 3., 7., 11., 19. /)
REAL:: Toplam_Akare,Toplam_Akup
INTEGER::SAYI=5
  INTERFACE
     SUBROUTINE Istatistik(X,Toplam_Xkare,Toplam_Xkup,N)
          IMPLICIT NONE
          INTEGER,INTENT(IN)::N
          REAL,INTENT(IN),DIMENSION(1:50)::X
          REAL,INTENT(OUT)::Toplam_Xkare,Toplam_Xkup
     END SUBROUTINE Istatistik
  END INTERFACE
CALL Istatistik(A,Toplam_xkare,Toplam_xkup,SAYI)
PRINT*, 'Kareler toplamı=',Toplam_Xkare
PRINT*, 'Küpler toplamı =',Toplam_XKup
END PROGRAM Ornek7

SUBROUTINE Istatistik(X, Toplam_Xkare, Toplam_Xkup, N)
IMPLICIT NONE
INTEGER,INTENT(IN)::N
REAL,INTENT(IN),DIMENSION(1:50)::X
REAL,INTENT(OUT)::Toplam_Xkare,Toplam_Xkup
INTEGER::i
REAL:: Gecici
Toplam_Xkare=0.0; Toplam_Xkup =0.0
DO i=1,N
     Gecici=X(I)*X(I)
     Toplam_Xkare= Toplam_Xkare + Gecici
     Toplam_Xkup = Toplam_Xkup  + X(I)*Gecici
END DO
END SUBROUTINE Istatistik
```

ÖRNEK 8: İkinci dereceden bir denklemin gerçek köklerini bulan, sanal kökleri olduğunda ise köklerin sanal olduğunu bildiren bir program yazınız. Programınız `INTERFACE` olarak tanımlanan iki `SUBROUTINE` içermesi istenmektedir: (1) ikinci dereceden denklemin katsayılarını okuyup, katsayıların veri girişi hatalarına karşın, kullanıcıyı uyaran bir alt program; (2) gerçek köklerin hesabını yapan ve ekrana yazdıran bir alt program.

Birinci alt program **`Etkilesim`** ve ikinci alt program da **`Coz`** ismiyle verilmektedir. `Etkilesim` alt programında `READ` deyiminde `IOSTAT`'dan yararlanılmaktadır. Yanlış veri girişi olduğunda (veri tipi, veri sayısı vs) `IOSTAT` pozitif tamsayı değeri almaktadır. Böylece `TAMAM` mantıksal değişkeni `.FALSE.` değeri alarak veri girişinin yanlış olduğunu çağıran programa bildirir.

```
PROGRAM Ornek8
IMPLICIT NONE
INTERFACE
     SUBROUTINE Etkilesim(A,B,C,TAMAM)
          IMPLICIT NONE
          REAL, INTENT(OUT) :: A
          REAL, INTENT(OUT) :: B
          REAL, INTENT(OUT) :: C
          LOGICAL, INTENT(OUT) :: TAMAM
     END SUBROUTINE Etkilesim

     SUBROUTINE Coz(E,F,G,Kok1,Kok2,Basarisiz)
          IMPLICIT NONE
          REAL, INTENT(IN)  :: E
          REAL, INTENT(IN)  :: F
          REAL, INTENT(IN)  :: G
          REAL, INTENT(OUT) :: Kok1
          REAL, INTENT(OUT) :: Kok2
          INTEGER, INTENT(INOUT) :: Basarisiz
     END SUBROUTINE Coz
END INTERFACE

REAL    :: P, Q, R, Kok1, Kok2
INTEGER :: Basarisiz=0
LOGICAL :: TAMAM=.TRUE.

     CALL Etkilesim(P,Q,R,TAMAM)
     IF (TAMAM) THEN
          CALL Coz(P,Q,R,Kok1,Kok2,Basarisiz)
          IF (Basarisiz == 1) THEN
               PRINT *,' Kökler sanal !'
          ELSE
               PRINT *,' Kökler  ',Kok1,'  ',Kok2
          ENDIF
     ELSE
          PRINT*,' Veri girişi hatası var'
     ENDIF
END PROGRAM Ornek8
```

```
SUBROUTINE Etkilesim(A,B,C,TAMAM)
IMPLICIT NONE
! ax^2+bx+c=0 denkleminin katsayıları
REAL,    INTENT(OUT) :: A, B, C
! Durum değişkeni, TAMAM=0 ise okuma Başarılı, /= ise başarısız
LOGICAL, INTENT(OUT) :: TAMAM
INTEGER              :: Durum=0
PRINT*,' A, B ve C Katsayılarını Giriniz '
READ(UNIT=*,FMT=*,IOSTAT=Durum) A, B, C
IF(Durum==0) THEN
     TAMAM=.TRUE.
ELSE
     TAMAM=.FALSE.
ENDIF
END SUBROUTINE Etkilesim

SUBROUTINE Coz(E,F,G,Kok1,Kok2,Basarisiz)
! ex^2+ fx+g =0 denkleminin kökleri
IMPLICIT NONE
REAL,    INTENT(IN)    :: E, F, G
REAL,    INTENT(OUT)   :: Kok1, Kok2 ! Gerçek kökler
INTEGER, INTENT(INOUT) :: Basarisiz  ! Sanal kökler
REAL                   :: Terim, A2
Terim = F*F - 4.*E*G
A2 = E*2.0
! Eğer Terim < 0 ise, Kökler sanaldır
IF(Terim < 0.0)THEN
     Basarisiz = 1
ELSE
     Terim = SQRT(Terim)
     Kok1 = (-F+Terim)/A2
     Kok2 = (-F-Terim)/A2
ENDIF
END SUBROUTINE Coz
```

7.8 STANDART DERLEYİCİ ARŞİV ALT PROGRAMLARI (FUNCTION VE SUBROUTINE'LER)

Arşiv fonksiyonları FORTRAN derleyicileri ile beraber kullanıcıya sunulan, çok hızlı bir şekilde sonuca ulaşan genellikler matematikte kullanılan bazı fonksiyonları içeren alt programlardır.

Arşiv fonksiyonlarının kullanım amaçlarını tanımlayan, örneğin karekök işlemi için `SQRT` ismi gibi, jenerik (tüm derleyicilerin kullandığı standart) isimleri vardır. Bundan başka, arşiv ismi, örneğin `DSQRT` gibi (çift hassasiyetli—Double precision—karekök alma), bu fonksiyonun esas amacını belirler. Özetle arşiv isimi tek hassasiyetli işlem yapılan durumlarda genellikle (bazı durumlar hariç) jenerik isim ile aynıdır. Çift hassasiyetli programlarda jenerik ismin başına `D` getirilerek kütüphane ismi elde edilir. Karmaşık sayılarla işlem yaparken, jenerik ismin başına `C` getirilir. Örneğin, `SIN(X)` fonksiyonu `REAL X`

argümanının sinüsünü verirken, `DSIN(X)` fonksiyonu `DOUBLE PRECISION` (veya `REAL*8`) olarak tanımlanmış `X` argümanının çift hassasiyetli olarak hesaplanmış sinüs değerini verir. Ayrıca `X`, `COMPLEX` olarak tanımlanmışsa, `CSIN(X)` fonksiyonu ile bu karmaşık değerin sinüsü hesaplanır. Ancak bazı istisnalar da söz konusudur; bazı matematiksel arşiv fonksiyonları ve kullanımlarına ilişkin bilgi Tablo 7.1'de verilmiştir. Bu nedenle fonksiyonlar ve kullanım şekilleri için, kullandığınız derleyicinin kullanım kılavuzuna başvurunuz.

7.9 SUBROUTINE-FUNCTION KARŞILAŞTIRILMASI

Genel olarak `SUBROUTINE` alt programları birden fazla çıktı değeri hesaplamak için kullanılır. Bu demektir ki, `FUNCTION` alt programları tarafından yapılabilen herhangi bir işlem, `SUBROUTINE` alt programları tarafından da yapılabilir.

- Altprogramları çağırma, `FUNCTION` alt programları ana programda adını vererek `SUBROUTINE`'lerde ise `CALL` deyimi kullanarak yapılır.

- `FUNCTION` ismine alt programda atama yapılırken `SUBROUTINE` ismi asla değişken gibi kullanılamaz.

- `FUNCTION` en az bir argümana sahip olması gerekir ve/veya değişken listesinde yer alan değişkenlerin bir kısmı veya tamamı `MODULE-USE` deyimleri ile alt programa aktarılabilir; `SUBROUTINE` 'lerin argüman listesi olması gerekmez.

- `FUNCTION` alt programlarının tipi (`REAL`, `INTEGER`, `COMPLEX`, `LOGICAL`, `CHARACTER`, vb) tip tanımlamaya tabidir; `SUBROUTINE`'ler veya isimleri herhangi bir veri tipinde tanımlanmaz.

Tablo 7.1: Bazı matematik arşiv alt programları ve kullanım şekilleri.

Jenerik İsmi	İşlemi	Kullanım İsmi	Argüman Tipi
ABS(A)	$\|a\|$	ABS(r) CABS(c) DABS(d) IABS(i)	Gerçek sayı Karmaşık sayı Çift hassasiyetli gerçek sayı Tamsayı
ACOS(X)	$\cos^{-1} x$	ACOS(r) DACOS(d)	Gerçek sayı Çift hassasiyetli gerçek sayı
ASIN(X)	$\sin^{-1} x$	ASIN(r) DASIN(d)	Gerçek sayı Çift hassasiyetli gerçek sayı
ATAN(X)	$\tan^{-1} x$	ATAN(r) DATAN(d)	Gerçek sayı Çift hassasiyetli gerçek sayı
ATAN2(Y,X)	$\tan^{-1} \frac{y}{x}$	ATAN(r2,r1) DATAN(d2,d1)	Gerçek sayı Çift hassasiyetli gerçek sayı
COS(X)	$\cos x$	COS(r) DCOS(d) CCOS(c)	Gerçek sayı Çift hassasiyetli gerçek sayı Karmaşık sayı
COSH(X)	$\cosh^{-1} x$	COSH(r) DCOSH(d)	Gerçek sayı Çift hassasiyetli gerçek sayı

DOT_PRODUCT(a,b)	**a•b**		Argüman tipleri belirleyici
EXP(X)	e^x	EXP(x) DEXP(x) CEXP(x)	Gerçek sayı Çift hassasiyetli gerçek sayı Karmaşık sayı
LOG(X)	$\ln x$	ALOG(x) DLOG(x) CLOG(x)	Gerçek sayı Çift hassasiyetli gerçek sayı Karmaşık sayı
LOG10(X)	$\log_{10} x$	ALOG10(x) DLOG10(x) CLOG10(x)	Gerçek sayı Çift hassasiyetli gerçek sayı Karmaşık sayı
MAX(a1,a2,..)		AMAX0(i1,i2,..) AMAX1(r1,r2,..) DMAX1(d1,d2,..) MAX0(i1,i2,..) MAX1(r1,r2,..)	Gerçek sayı Gerçek sayı Çift hassasiyetli gerçek sayı Tamsayı Tamsayı
MIN(a1,a2,..)		AMIN0(i1,i2,..) AMIN1(r1,r2,..) DMIN1(d1,d2,..) MIN0(i1,i2,..) MIN1(r1,r2,..)	Gerçek sayı Gerçek sayı Çift hassasiyetli gerçek sayı Tamsayı Tamsayı
SIN(X)	$\sin x$	SIN(r) DSIN(d) CSIN(c)	Gerçek sayı Çift hassasiyetli gerçek sayı Karmaşık sayı
SINH(X)	$\sinh x$	SINH(r) DSINH(d)	Gerçek sayı Çift hassasiyetli gerçek sayı
SQRT(X)	$\sqrt{x}$	SQRT(r) DSQRT(d) CSQRT(c)	Gerçek sayı Çift hassasiyetli gerçek sayı Karmaşık sayı
TAN(X)	$\tan x$	TAN(r) DTAN(d)	Gerçek sayı Çift hassasiyetli gerçek sayı
TANH(X)	$\tanh x$	TANH(r) DTANH(d)	Gerçek sayı Çift hassasiyetli gerçek sayı

ALIŞTIRMALAR

7.1 Aşağıdaki `FUNCTION` ve `SUBROUTINE` isim tanımlarında, varsa hataları, tespit ediniz.

```
(a) REAL FUNCTION Bis(n,x)
(b) FUNCTION Matris(2*n,a,b)
(c) SUBROUTINE Tarama(a,c,y,L(n))
(d) SUBROUTINE ENBüyük(N,X,Y,EB)
(e) FUNCTION (a,i,j)
(f) SUBROUTINE Vektor(K,I(5),M,L)
(g) FUNCTION Borçlu(m,a,x)
(h) COMPLEX SUBROUTINE ZCZ(z1,z2,z)
(i) FUNCTION*8 Funcf(X,Y,Z)
(j) LOGICAL FUNCTION OkeyMi(L)
(k) FUNCTION NOTLAR a,b,c
(l) SUBROUTINE*l6 Kompleks(A,B,C,N)
(m) SUBROUTINE (U,V,W)
(n) LOGICAL FUNCTION, Son(a,b+c,e*f)
```

7.2 Aşağıdaki işlemleri yapan `FUNCTION` alt programları yazınız.

(a) $A(x) = \dfrac{e^{-\alpha x}}{1+x^2} + \log_2(3+\sin \pi x)$

(b) $B(x,y) = \dfrac{\log_5(1+x/y)}{e^x + e^{y-x}}$

(c) $C(x,y) = \displaystyle\sum_{n=0}^{30} \frac{n!}{1+n^2x^n + y^{n/4}}$

(d) $D(x,y,z) = \sqrt{\dfrac{x}{y}}\sin\sqrt{\dfrac{z}{y}} - \sqrt{\dfrac{y}{x}}\cos\sqrt{\dfrac{x}{x}}$

(e) $E(x,y,z) = \displaystyle\prod_{n=1}^{12}\left(\frac{x-n}{x+n}\right)\left(\frac{y-2n}{y+2n}\right)\left(\frac{z-3n}{z+3n}\right)$

7.3 Uzunluğu `M` olan bir `A` indisli değişkeninin değerlerinin ortalamasını hesaplayan `ORTALA` isimli `FUNCTION` alt programı hazırlayınız.

7.4 Herhangi bir $f(x)$ fonksiyonunun $[a, b]$ aralığında x ile değişimini $\Delta x = (b-a)/M$ (M bir tamsayıdır) artırımları kullanarak tablo haline getiren bir program yazınız. Bahsedilen işlemler bir `SUBROUTINE` içinde yer alacak, $f(x)$ fonksiyonu da `FUNCTION` alt programı olarak tanımlanacaktır.

7.5 Bir `REAL x` (`x`>0) sayısının herhangi bir `y INTEGER` tabanına göre logaritmasını ($\log_y x$) hesaplayan bir harici fonksiyon (`FUNCTION LOGY(X,Y)` isimli) hazırlayınız.

7.6 Literatürde Hata fonksiyonu (*Error Function*) olarak bilinen $erf(x)$ fonksiyonu

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_{x}^{\infty} e^{-u^2} du$$

olarak tanımlanıyor. Fakat bu integralin hesabı yerine, aşağıda verilen formülün programlanarak, herhangi bir *x* değeri için hata fonksiyonunu hesaplanmak istenmektedir:

$$erf(x) = 1 - e^{-x^2} \sum_{n=1}^{5} a_n t^n$$

Yukarıdaki formülde kullanılan sabitler aşağıda verilmiştir:

$$t = \frac{1}{1+px}, \quad p = 0.3275911, \; a_1 = 0.2548296, \; a_2 = -0.284497, \; a_3 = 1.421414,$$

$$a_4 = -1.453152, \quad a_5 = 1.061405$$

Hata fonksiyonunu, yukarıdaki bilgiler ışığı altında, hesaplayan bir `FUNCTION ERF(X)` yazınız ve bu fonksiyonu test ediniz. Bu test için *erf*(l)=0.84270, *erf*(1.25)=0.9229, *erf*(2)=0.99532 olarak bildirilmektedir.

7.7 Bir $f(x)$ fonksiyonunun herhangi bir *x*=*a* noktasındaki birinci dereceden türevi

$$f'(x) \cong \frac{f(x+h) - f(x-h)}{2h}$$

olarak veriliyor. Argümanları *a* ve *h* olan, birinci dereceden türevi hesaplayan bir `SUBROUTINE` alt programı yazınız. *NOT*: $f(x)$ fonksiyonu `EXTERNAL` olarak tanımlanacaktır.

7.8 Boyutu *n* olan *x* ve *y* indisli değişkenleri için, aşağıdaki ifade hesaplanmak isteniyor.

$$\frac{\sqrt{x_1 y_1 + x_2 y_2 + \cdots + x_n y_n}}{\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}\,\sqrt{y_1^2 + y_2^2 + \cdots + y_n^2}}$$

İfadeyi bir bilgisayar programı vasıtasıyla hesaplamak için $a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$ işlemini yapan `FUNCTION Vek(a,b)` isimli alt programı yazınız; programınızda bu fonksiyondan faydalanınız. İ*PUCU:* Fonksiyon tanımlandıktan sonra ifadeyi `Vec(x,y)/(Vek(x,x)*Vek(y,y)` ile hesaplayabiliriz.

7.9 Bir tamsayı indisli değişkeni olan `KOD(i,j)` ekrana matris şeklinde yazdırmak için `SUBROUTINE PRINT` isimli bir alt program yazınız.

7.10 Bir tamsayı indisli değişkeni olan `KOD(i,j)` ekrana matris şeklinde yazdırmak için `SUBROUTINE PRINT` isimli bir alt program yazınız.

7.11 `X(N,N,N)` indisli değişkeninin elemanlarından mutlak değerce en büyüğünü veren `FUNCTION Max_Indisli` isimli bir alt program yazınız

7.12 Tanımlayacağınız $R = \sqrt{a^2 + b^2 + c^2}$ fonksiyonu aracılığıyla aşağıdaki ifadeleri tek bir ana programda hesaplayan bir program yazınız.

$$A = \frac{x}{\sqrt{x^2 + y^2 + z^2}} \qquad B = \sqrt{x^4 + y^4 + z^4} \qquad C = \frac{1}{\sqrt{4x^2 + 9y^2 + 16z^2}}$$

7.13 Aşağıdaki programların çıktılarını bulunuz.

(a)
```
PROGRAM Problem_a
IMPLICIT NONE
REAL :: x=2., y=4., z=5.
REAL :: d, Fonks
d=x+y+z
d=d+Fonks(x,y)
PRINT *, 'd=',d
END PROGRAM Problem_a

REAL FUNCTION Fonks(a,b)
IMPLICIT NONE
REAL, INTENT(IN) :: a, b
   Fonks=a*a+b/a
END FUNCTION Fonks
```

(b)
```
PROGRAM Problem_b
IMPLICIT NONE
REAL :: x=2., y=3., h=1.e-5
REAL :: Fxy, h2, fx, fy
h2=2.*h
fx=(fxy(x+h,y)-fxy(x-h,y))/h2
fy=(fxy(x,y+h)-fxy(x,y-h))/h2
PRINT 100, 'fx=',fx,'fy=',fy
100 FORMAT(3(1x,A,2x,F10.5/))
END PROGRAM Problem_b

REAL FUNCTION Fxy(a,b)
IMPLICIT NONE
REAL, INTENT(IN) :: a, b
   Fxy=SIN(a*b)
END FUNCTION Fxy
```

(c)
```
PROGRAM Problem_c
IMPLICIT NONE
REAL :: DX=0.5, X
REAL :: Fonk_A, Fonk_B
INTEGER :: n=0
X=DX
DO
     n=n+1
     PRINT 100, X, Fonk_A(X), Fonk_B(X)
     X=X+DX
     IF(n>11) EXIT
END DO
100 FORMAT(3(1x,F10.5))
END PROGRAM Problem_c
```

```
REAL FUNCTION Fonk_A(u)
IMPLICIT NONE
REAL, INTENT(IN) :: u
   Fonk_A=u*u-1.
END FUNCTION Fonk_A

REAL FUNCTION Fonk_B(u)
IMPLICIT NONE
REAL, INTENT(IN) :: u
   Fonk_B=u*u-3.*u+2
END FUNCTION Fonk_B
```

(d)

```
PROGRAM Problem_d
IMPLICIT NONE
REAL :: x=-1., y=0.
REAL :: x0, y0, FA, FB, FC
DO
     IF(FA(x)<FB(y)) THEN
          x0=FB(x)
          y0=FA(x0+1.)
     ELSE
         x0=FA(x+1.)
         y0=FB(x0+1.)
     ENDIF
     PRINT 100, x0,y0,FC(x0,y0)
     x=x+1.
     y=y+2.
     IF(x+y>9) EXIT
END DO
100 FORMAT(5x,'(',f7.4,',',f7.4,')=',f9.4)
END PROGRAM Problem_d
!
REAL FUNCTION FA(x)
IMPLICIT NONE
REAL, INTENT(IN) :: x
    FA=x*x+3.*x+2.
END FUNCTION FA
!
REAL FUNCTION FB(x)
IMPLICIT NONE
REAL, INTENT(IN) :: x
    FB=x+3.
END FUNCTION FB
!
REAL FUNCTION FC(x,y)
IMPLICIT NONE
REAL, INTENT(IN) :: x,y
    FC=(x+y)/(2.*x-y)
END FUNCTION FC
```

(e)
```
PROGRAM Problem_e
IMPLICIT NONE
REAL, DIMENSION(3) :: c=(/1.,3.,4./), d
INTEGER :: n=3
REAL :: y, y2, asn
y =asn(n,c)
d(1)=c(1)
d(2)=c(2)+1.5
y2=asn(n,d)
PRINT 100, y,y2
100 FORMAT(2x,f8.5,1x,f9.5)
END PROGRAM Problem_e
!
REAL FUNCTION asn(m,x)
IMPLICIT NONE
INTEGER, INTENT(IN) :: m
REAL, DIMENSION(m), INTENT(IN) :: x
INTEGER :: i
asn=0.0
DO i=1,m
  asn=asn+x(i)*x(i)
END DO
END FUNCTION asn
```

(f)
```
MODULE modul

CONTAINS
   REAL FUNCTION p1(u,v)
   IMPLICIT NONE
   REAL, INTENT(IN) :: u, v
     p1=u*u-v*v
   END FUNCTION p1

   REAL FUNCTION p2(u,v)
   IMPLICIT NONE
   REAL, INTENT(IN) :: u, v
     p2=u*u+v*v
   END FUNCTION p2
END MODULE modul
!
PROGRAM Program_f
USE modul
IMPLICIT NONE
REAL :: t=1.0, a=2.0, y
INTEGER :: i=1
DO
   IF(t>=a) THEN
       y=p1(t,a)
       t=t+2.0
   ELSE
       y=p2(t,a)
```

```
        t=t+1.0
        a=a-1.0
    ENDIF
    PRINT 100, 't=',t,'y=',y
    IF(i==3) EXIT
    i=i+1
END DO
100 FORMAT(2x,2(A2,1x,f8.5))
END PROGRAM Program_f
```

(g)

```
MODULE modul

CONTAINS
        SUBROUTINE abc(x,y,z,a,b,c)
        IMPLICIT NONE
        REAL, INTENT(IN) :: x, y, z
        REAL, INTENT(OUT):: a,b,c
        a=(x+y+z)/3.0
        b=SQRT(x*x+y*y+z*z)
        c=(a-b)/(a+b)
        END SUBROUTINE abc
END MODULE modul
!
PROGRAM g
USE modul
IMPLICIT NONE
REAL :: a=4.0, b=2.0, c=0.0, d=4.0
REAL :: u, v, w
CALL abc(a,b,c,u,v,w)
     PRINT 100, u,v,w
CALL abc(a,b,d,u,v,w)
     PRINT 100, u,v,w
CALL abc(a,c,d,u,v,w)
     PRINT 100, u,v,w
100 FORMAT(3x,3(ES10.3,1x))
END PROGRAM g
```

(h)

```
MODULE modul

CONTAINS
     SUBROUTINE hesap(m,u,a,b)
     IMPLICIT NONE
     INTEGER, INTENT(IN) :: m
     REAL, DIMENSION(M), INTENT(IN) :: u
     REAL, INTENT(OUT)   :: a, b
     REAL    :: gecici
     INTEGER :: i
        DO i=1,m
          gecici=u(i)*u(i)
          a=a+gecici
          b=b+gecici*u(i)
```

```
         END DO
      END SUBROUTINE hesap
END MODULE modul
!
PROGRAM Program_h
USE modul
IMPLICIT NONE
REAL, DIMENSION(4) :: a=(/ 1., 3., -2., 6. /)
REAL, DIMENSION(5) :: b=(/-2., 4.,  1., 5. , 8. /)
INTEGER :: na=4, nb=5
REAL :: x, y
CALL hesap(na,a,x,y)
PRINT *, 'a icin hesaplar',x,y
CALL hesap(nb,b,x,y)
PRINT *, 'b icin hesaplar',x,y
END PROGRAM Program_h
```

(i)

```
PROGRAM Program_i
IMPLICIT NONE
INTEGER :: j
REAL :: X
REAL , DIMENSION(10) :: Y
!
INTERFACE
  ELEMENTAL REAL FUNCTION ETOX(X)
  IMPLICIT NONE
  REAL , INTENT(IN) :: X
  END FUNCTION ETOX
END INTERFACE
!
  X=1.0
  DO j=1,10
    Y(j)=j
  END DO
  PRINT 100, Y
  X=ETOX(X)
  PRINT 100,X
  Y=ETOX(Y)
  PRINT 100,Y
  100 FORMAT(1x,10(2x,f9.3/))
END PROGRAM Program_i
!
ELEMENTAL REAL FUNCTION ETOX(X)
IMPLICIT NONE
REAL , INTENT(IN) :: X
REAL :: TERIM
INTEGER :: NTERIM
REAL , PARAMETER :: TOL =1.0E-6
  ETOX=1.0
  TERIM=1.0
```

```
  NTERIM=0
  DO
    NTERIM=NTERIM+1
    TERIM=(X/NTERIM)*TERIM
    ETOX=ETOX+TERIM
    IF (TERIM<=TOL) EXIT
  END DO
END FUNCTION ETOX

(j)  PROGRAM Program_j
     IMPLICIT NONE
     REAL :: Hi , Hr , HL , HH , HA , Xl
     REAL :: Xh , Xm , D
     REAL , PARAMETER :: Tol=1.0E-6
     HH = HL*2.0
     Xl = F( HL, Hr, D)
     Xh = F( HH, Hr, D)
     DO WHILE ((XL*XH)  >= 0.0)
          HH = HH*2.0
          XH = F(HH,HR,D)
     END DO
     DO
          HA=(HL+HH)*0.5
          XM=F(HA,HR,D)
          IF((XL*XM)<0.0)THEN
               XH=XM
               HH=HA
          ELSE
               XL=XM
               HL=HA
          ENDIF
          IF(ABS(HH-HL)<= TOL)EXIT
     END DO
     PRINT*, HL,' ve ',HH,

CONTAINS
  REAL FUNCTION F(A,B,C)
  IMPLICIT NONE
  REAL , INTENT (IN) :: A
  REAL , INTENT (IN) :: B
  REAL , INTENT (IN) :: C
     F=A*(1.0-0.8*EXP(-0.6*C/A))-B
  END FUNCTION F

END PROGRAM Program_j
```

BÖLÜM 8
ALFA SAYISALLARIN UYGULAMALARI

Alfa sayısallardan genellikle kelime-işlemci ve grafik programlarında olarak faydalanılır. Bunun yanı sıra sayısal çıktıların düzenlenmesinde de geniş çapta yararlanılır. Hatırlanacağı üzere, alfa sayısal değişkenler program başında **CHARACTER** tip belirleyicisi ile aşağıdaki gibi tanımlanırlar:

```
CHARACTER(LEN=14) :: Ad, Soyad
CHARACTER(LEN=4), DIMENSION(0:12) :: Gaz
CHARACTER(LEN=1), DIMENSION(6) :: Sembol
CHARACTER(LEN=5), DIMENSION(21):: no
```

Bu değişkenlere daha sonra alfa sayısal değerler atanabilir. Bu değişkenlerden `no`, `Gaz` ve `Sembol` indisli değişkenler olarak kullanılmışlardır. Örneğin, bu değişkenlerin bazılarına yapılan atamalar ile bellekteki yerleşimi aşağıda verilmektedir:

Atanan Değer	Bellekteki Yerleşimi				
No(1)='2601A'	2	6	0	1	A
No(2)='2601B'	2	6	0	1	B
Gaz(0)='ETAN'	E	T	A	N	
Gaz(1)='AZOT'	A	Z	0	T	
Gaz(4)='HE'	H	E			
Sembol(1)='*'	*				
Sembol(3)='!'	!				

Atanan karakter değerler sırasıyla, alfa sayısalın uzunluğuna göre, soldan sağa doğru bellekteki adreslerine yerleştirilirler.

Alfa sayısallarla kıyaslama yapma işlemi normal sayısallara uygulanan tekniklere benzemektedir: örneğin, 100 kişilik bir gurubun içinde ismi AHMET ve soyadı TURK olan birisinin tespitini veren bir program örneği verilmektedir.

```
PROGRAM isim_arama
IMPLICIT NONE
INTEGER :: i
CHARACTER(LEN=14), DIMENSION(100):: Ad, Soyad
OPEN(5,FILE='ISIMLER',STATUS='OLD')
READ(5,*) (Ad(i), Soyad(i), i=1,100)
DO i=1,100
     IF((Ad(i)=='AHMET').AND.(Soyad(i)=='TURK')) THEN
         PRINT*, 'KAYITLARDA AHMET TURK MEVCUTTUR.'
     ENDIF
END DO
END PROGRAM isim_arama
```

Bu programda 100 kişinin ad ve soyadı indisli değişkenlerde depolanmıştır; bunlardan ismi AHMET ve soyadı TURK olan kaydın mevcudiyeti yukarıdaki şekilde araştırılabilir.

8.1 ALFA SAYISALLARIN KIYASLANMASI

Sembollerin 'A', 'B',...,'Z' ikilik düzendeki kodlan, o şekilde ayarlanmıştır ki bunların değerleri sembol sırasını takip eder; yani 'A'nın değeri 'B'nin değerinden ve 'C'nin değeri 'B'nin değerinden v.s daha büyüktür. Bu ikilik kodlama sistemine ASCII kodu denir. Küçük harflerin ('a', 'b',…, 'z') kodları büyük harflerden büyüktür. Bu nedenle karakter değişkenlerin değerleri mantıksal IF ile kıyaslanabilir. Örnekler,

```
('A'<'D')            TRUE
('AA'<'A')           FALSE
('ABC'<='ABE')       TRUE
('a'>'A')            TRUE
('E'<'c')            TRUE
```

Yukarıdaki örneklerden ikincisinde 'A' nın bellek değeri 'A', yani A artı boşluk olarak kabul edilir ve boşluğun ikilik kodu A'dan daha küçük olduğundan önerme yanlıştır. Üçüncü önermede ilk iki karakter aynı olmasına rağmen üçüncü karakterlerdeki farklılık kıyaslamaya esas teşkil etmektedir.

Alfa sayısallar sadece harflerden oluşmayabilir; rakamlar da içerebilirler. Örneğin

```
('234'=='234')       TRUE
('234'=='2345')      FALSE
```

olacaktır.

Alfa sayısallarda kullanılan harfler ASCII kodlama sistemine tabidir ve kıyaslama bu kodlar üzerinden yapılır. Örneğin, 'A' karakterinin kod numarası 65, 'B'nin 66'dır. Bu nedenle, 'B'>'A'dır. Diğer taraftan, 'a'nın kodu 97 olup 'a'>'A' olmaktadır. 'A'dan 'Z'e harflerin kodları alfabetik sıralamayı takip etmektedir; yani 65, 66, … '0'dan '9'a olan rakamlar kendi değerleri ile kodlanmıştır. Bunların dışındaki alfa sayısallar, örneğin 'Ali', kodlanmamıştır.

Bir karakterden uzun alfa sayısalların kıyaslamaları karakterlerin karşılaştırılması ile gerçekleştirilir. Alfa sayısalların ilk karakterleri kıyaslamaya tabi tutulur. İlk karakterler aynı ise, ikinci karakterlerin kıyaslanmasına geçilir ve bu işlem eşitlik bozuluncaya kadar devam ettirilir; örneğin 'AAAA' ile 'AACA' kıyaslamasında eşitlik üçüncü karakterde bozulmuştur; dolayısıyla, 'AAAA'<'AACA'dır.

Alfa sayısalların uzunlukları farklı ise, kıyaslama ilk karakterlerle başlanır; farklı bir karakter ile karşılaşıncaya kadar karakter-karakter kıyaslama yapılır. Alfa sayısalın birinin sonuna kadar olan karakterler aynı ise, iki alfa sayısaldan diğerinin daha büyük olduğu kabul edilir. Örneğin, 'AB'>'AAAA' ve 'AAAA'>'AAA'.

Bazı bilgisayarlar, özellikle IBM firması, EBCDIC (*Extended Binary Coded Decimal Interchange Code*) kodlama sistemi kullanır. Yukarıda bahsedilen kıyaslama makinenin kullandığı kodlamaya bağlı olduğundan, kıyaslama algoritması yetersiz, eksik veya yanlış

çalışmasına neden olabilmektedir. Bu nedenle, Fortran programlama dilinde, kullanılan farklı kodlama sistemlerinden bağımsız olması amacıyla, arşiv programları arasına dört adet *mantıksal kıyaslama fonksiyonu* mevcuttur: **`LLT`** (*lexically less than-sözcüksel olarak küçük*), **`LLE`** (*lexically less than or equal to-sözcüksel olarak küçük veya eşit*), **`LGT`** (*lexically greater than-sözcüksel olarak büyük*) ve **`LGE`** (*lexically greater than or equal to-sözcüksel olarak büyük eşit*). Bu fonksiyonlar <, <=, > ve >= ilişkisel operatörlerinin birebir karşılığıdır; ancak yukarıda bahsi geçen sözcüksel kıyaslama fonksiyonları kullanılan bilgisayar ne olursa olsun, kıyaslamanın ASCII kodlama sistemine göre yapılmasına neden olur.

Aşağıdaki örnekte `alfa1` ve `alfa2` alfa sayısal değişkenleri `<` operatörü ve **`LLT`** ile kıyaslanmaktadır. `Sonuc1` işlemciden işlemciye değişirken, `sonuc2` işlemci ne olursa olsun daima `TRUE` olacaktır.

```
LOGICAL :: sonuc1, sonuc2
CHARACTER(LEN=6) :: alfa1, alfa2
alfa1='A1'
alfa2='a1'
sonuc1=alfa1<alfa2
sonuc2=LLT(alfa1, alfa2)  ! => True
```

8.2 ALT ALFA SAYISALLAR

Bir alfa sayısal birkaç karakterden birkaç bin karakter uzunluğunda olabilir. Böyle çok uzun alfa sayısallarda, alfa sayısalın sadece belirli kesimlerine erişmek gerekebilir. Bu şekilde alfa sayısalın sadece belirli bir kısmını içeren alfa sayısala alt alfa sayısal (*substring*) denir. Alt alfa sayısallar FORTRAN'ın aşağıdaki özelliğinin kullanımı ile oluşturulabilir.

```
Alfa_sayısal(nsol:nsağ)
```

burada `alfa_sayısal` herhangi bir alfa sayısal değişkeninin ismi, `nsol` ilk karakter veya en soldaki karakterin konumunu veren tamsayıdır; kullanılmadığında l değerini alır; `nsag` ise alt alfa sayısalın son karakter veya en sağdaki karakterin konumunu veren tamsayıdır. Bunun için de herhangi bir değer kullanılmadığında alfa sayısalın en son konumunu esas alır. Örneğin, `XY` alfa sayısalının değerine göre oluşturulan alt alfa sayısallar ve değerleri aşağıda verilmektedir:

```
CHARACTER(LEN=15) :: XY
XY='1234567890ABCDE'         !   Çıktı Değerleri
PRINT*, XY(1:2)              !    '12'
PRINT*, XY(3:7)              !    '34567'
PRINT*, XY(:5)               !    '12345'
PRINT*, XY(10:)              !    '0ABCDE'
PRINT*, XY(11:11)            !    'A'
PRINT*, XY(9:10)             !    '0A'
```

8.3 KÜTÜK SONUNUN BULUNMASI

Sıklıkla bir kütükteki belirsiz sayıda kayıtın programa okunması ve bu bilgilerin bir algoritma uyarınca işlenmesi gerekmektedir. Bu durumda kayıt sayısının bilinmemesi durumunda, kütük sonuna ulaşıldığında uygun programlama yapılmamışsa, *kütük okuma hatası* (bir çalıştırma hatasıdır) ile karşılaşılabilir. Bu problemden sakınmak için kütük sonuna ulaşılıp ulaşılmadığını kontrol etmek, ulaşıldığında da hata mesajına sebebiyet vermeden program akış yönünü değiştirmek gerekir.

Bu amaç için kullanılan bir teknik, `READ` deyiminin kullanıma tahsis edilen `IOSTAT` parametresini içerir. `IOSTAT` parametresi, `IOSTAT=durum` şeklinde kullanılır (`durum=0` ise girdi/çıktı işlemi başarılı, `-1` ise kütük sonunu `-2` ise kayıt sonunu, pozitif bir sayısı ise başarısız okuma olduğu anlamına gelmektedir) ve `EOF` (End-Of-File) mantıksal sabiti, kütük sonu işaretine rastlandığında, program akış yönünün `IOSTAT` ile değiştirilmesine olanak sağlar.

ÖRNEK 1: Bir kütükteki sayısı bilinmeyen kayıtlar şahıslara ait şu bilgileri içermektedir: `Isim` (A15), `Soyad` (A10), `Adres` (A70), `TelNo` (A8). Bu kayıtları okuyarak kütükte kaç kişinin olduğunu tespit etmek için bir yazınız.

Yukarıdaki açıklamalar ışığı altında `READ` deyiminde `IOSTAT` parametresinden faydalanarak aşağıdaki şekilde bir program yazmak mümkündür.

```
PROGRAM Ornek1
IMPLICIT NONE
! Kayıt Okuma Ve Sayısını Bulma Programı
CHARACTER(LEN=15) :: Isim
CHARACTER(LEN=10) :: Soyad
CHARACTER(LEN=70) :: Adres
CHARACTER(LEN=8)  :: TelNo
INTEGER :: durum, n=0       ! n kayıt Sayısı
OPEN(5,FILE='KAYIT.DAT',STATUS='OLD')
DO
     READ(5,100,IOSTAT=durum) Isim,Soyad,Adres,TelNo
     n=n+1
     IF(durum/=-1) EXIT
END DO
PRINT*, 'Kütük sonuna ulaşıldı... '
100 FORMAT(A15,1X,A10,1X,A70,1X,A8)
END PROGRAM Ornek1
```

şeklinde yazılabilir. Burada `eof` bir mantıksal arşiv fonksiyonu olup kütük sonuna ulaşıldığında TRUE değerini alır; veya yukarıdaki programa alternatif olarak

```
DO WHILE(eof) ! eof=.FALSE. olduğu sürece döngü işlemlerini yapar
   READ(5,100) Isim, Soyad, Adres, TelNo
   n=n+1
END DO
```

`DO-WHILE` dönüsü kullanabiliriz.

READ deyimi bir sayaçlı döngü içine yerleştirilerek okunan kayıtlar sayılır; kütük sonuna ulaşıldığında döngüden EXIT deyimi ile çıkılır; döngü sonundaki "Kütük sonuna ulaşıldı…" mesajını verir..

8.4 ALFA SAYISAL ARŞİV FONKSİYONLARI

Standart Fortran derleyicileri, alfa sayısalların kullanımına uygun olarak hazırlanmış arşiv fonksiyonları içermektedir. Bu fonksiyonların en sık kullanılanları ve işlevleri Tablo 8.1'de verilmiştir.

Tablo 8.1 Alfa sayısallarla kullanılan arşiv fonksiyonları.

Fonksiyon	İşlevi
ACHAR(i)	ASCII kodlama sistemindeki *i*.ci karakter
ADJUSTL(*alfa_sayısal*)	Alfa sayısalı soldan hizala
ADJUSTR(*alfa_sayısal*)	Alfa sayısalı sağdan hizala
CHAR(i)	İşlemci sistemindeki *i*.ci karakter
IACHAR(*karakter*)	Girilen karakterin ASCII sistemindeki konumu (integer)
ICHAR(*karakter*)	Girilen karakterin işlemci sistemindeki konumu (integer)
INDEX(*alfa,alt_alfa*)	Alt alfa sayısal değerin alfa içindeki başlangıç konumu
LEN(*alfa_sayısal*)	Alfa sayısalın karakter uzunluğu (integer)
LEN_TRIM(alfa_sayısal)	Alfa sayısalın sonlarındaki boşluklar çıkarıldıktan sonraki karakter uzunluğu (integer)
LGE(*alfa_1,alfa_2*)	Sözcüksel kıyaslama fonksiyonu—Mantıksal büyük eşit (alfa1≥ alfa2 ⇒ .TRUE. veya .FALSE. değerleri alır.
LGT(*alfa_1,alfa_2*)	Sözcüksel kıyaslama fonksiyonu—Mantıksal büyük (alfa1>alfa2 ⇒ .TRUE. veya .FALSE. değerleri alır.
LLE(*alfa_1,alfa_2*)	Sözcüksel kıyaslama fonksiyonu—Mantıksal küçük eşit (alfa1≤ alfa2 ⇒ .TRUE. veya .FALSE. değerleri alır.
LLT(*alfa_1,alfa_2*)	Sözcüksel kıyaslama fonksiyonu—Mantıksal küçük (alfa1<alfa2 ⇒ .TRUE. veya .FALSE. değerleri alır.
REPEAT(*alfa_sayısal,i*)	Alfa_sayısal değerini i kez tekrarla
SCAN(*alfa_s,set*)	Bir alfa sayısal set içinde alfa_s değerini taramak
TRIM(alfa_sayısal)	Alfa sayısalın varsa sonlarındaki boşlukları kesmek/çıkarmak
VERIFY(*alfa_s,set*)	Alfa sayısal set içinde alfa_s varlığını doğrulamak

Bu arşiv programlarından, örneğin **LEN** fonksiyonu alfa sayısal değişkenin uzunluğunu (değişkende kullanılan karakter sayısı olarak) verir. Kullanımı LEN (alfa sayısal değişken) şeklinde olup sonucu bir tamsayıdır. Bu fonksiyonun kullanımına ilişkin bazı örnekler aşağıda verilmektedir:

```
PROGRAM ornek
IMPLICIT NONE
INTEGER :: i1, i2, j1, j2
CHARACTER(LEN= 9):: SAYI='123456789'
CHARACTER(LEN=12):: HARF='ABCDEFGHIJKL'
i1=LEN(SAYI);       j1=LEN(HARF)
i2=LEN(SAYI(3:7));  j2=LEN(HARF(:10))
PRINT*, i1,j1,i2,j2
END PROGRAM Ornek    ! Program çıktısı  9   12  5   10
```

Diğer taraftan, bir alfa sayısalın içindeki herhangi bir alt alfa sayısalın konumunu tespit etmeye yarayan **INDEX** fonksiyonu

```
INDEX(alfa_sayısal:alt_alfa_sayısal)
```

şeklinde kullanılır. Örneğin

```
CHARACTER(LEN=15) :: XY
INTEGER :: i, j, k
XY='1234567890ABCDE'
i=INDEX(XY,'A')
j=INDEX(XY,'678')
k=INDEX(XY,'3')
PRINT*, i, j, k
```

Program parçasında `i`, `j` ve `k`'nın aldığı değerler sırasıyla `10`, `6` ve `3` olmaktadır. Bu değerler de alt alfa sayısalın başlangıç konumunu verir. Eğer alt alfa sayısal esas alfa sayısalda mevcut değilse, **INDEX** değeri `0` olur.

CHAR fonksiyonu bir tamsayı argümanı olan bir fonksiyondur ve çıktısı ASCII kodlama sistemine göre bu kodlama sayısına (tamsayıya) karşılık gelen karakterdir. Örneğin, `CHAR(65)='A'`dır.

ICHAR fonksiyonunun argümanı bir karakterden oluşan alfa sayısaldır ve çıktısı tamsayı olup, bu karakterin ASCII kodlama sistemindeki tamsayı kodudur. Örneğin, **ICHAR**`('A')=65`'dir.

ACHAR ve **IACHAR** sırasıyla, tamamıyla, `CHAR` ve `IACHAR` ile aynı işleve sahiptir. Ancak bu fonksiyonlar, işlemci ne olursa olsun (farklı işlemcilerden etkilenmeden), karakterlerin ASCII kodlarını getirir.



*Yazacağınız program farklı bilgisayarlarda ve işlemcilerde çalıştı-rılacak ise, kıyaslamalarda sözcüksel kıyaslama fonksiyonlarını (***LLT***,* **LLE***,* **LGT***,* **LGE***) ve* **ACHAR** *ile* **IACHAR** *fonksiyonlarını kullanın!*

Alfa sayısallarla işleme müsaade edilen tek operasyon *birleştirme* işlemidir. Ancak birleştirme işlemi "+" işareti ile yapılmaz! S_1 ve S_2 gibi iki alfa sayısal sırasıyla n_1 ve n_2 uzunluklarında olsun, bunlar daha sonra `//` operatörüyle aşağıda verildiği gibi birleştirilebilir.

$$S_1//S_2$$

Birleştirilmiş alfa sayısalın uzunluğu, artık, $n_1 + n_2$ olur. Örneğin `AHMET` ve `TURK` isimlerinin tek bir alfa sayısal değişken altında birleştirilmesi aşağıdaki gibi gerçekleştirilebilir:

```
PROGRAM AlfaSayisal
IMPLICIT NONE
CHARACTER(LEN=20):: Ad, Soyad
CHARACTER(LEN= 1)  :: Nokta='.', &
            Virgul=',', Bos=' '
Ad    ='AHMET'
Soyad ='TURK'
Nokta ='.'
Virgul=','
Bos   =' '
PRINT*, Ad//Virgul//Bos//Soyad//Nokta
END PROGRAM AlfaSayisal
```

Program çıktısı

```
         1         2         3         4
1234567890123456789012345678901234567890123
AHMET               , TURK                .
```

olarak elde edilir. Çıktıda oluşan boşluklar `Ad` ve `Soyad`'ın 20 karakter uzunluğunda tanımlanmış olmasından kaynaklanmaktadır. Aşağıdaki şekilde verilen düzenleme ile

```
PROGRAM AlfaSayisal
IMPLICIT NONE
CHARACTER(LEN=20) :: Ad, Soyad
CHARACTER(LEN=1)  :: Nokta='.', &
   Virgul=',', Bos=' '
INTEGER :: i, j
Ad   ='AHMET'
Soyad='TURK'
i=INDEX(Ad   ,Bos)
j=INDEX(Soyad,Bos)
PRINT*, Ad(1:i-1)//Virgul//Bos//Soyad(1:j-1)//Nokta
END PROGRAM
```

program çıktısı

```
         1
123456789012345
AHMET, TURK.
```

olarak elde edilir; yani boşluk yeri tespit edildikten sonra, alfa sayısalların sadece boşluğa kadar olan kısmı birleştirme işlemine katılır.

`LEN_TRIM(alfa sayısal)`, alfa sayısalın sonundaki boşlukları çıkardıktan sonra alfa sayısalın karakter uzunluğunu verir.

```
CHARACTER(LEN=20) :: isim='Ahmet'
PRINT*, LEN(isim)
PRINT*, LEN_TRIM(isim)
```

Program parçasının çıktısı sırasıyla `20` ve `5` olmaktadır. Her ne kadar alfa sayısalın tanımlama ve atamasının yapıldığı ilk satırda, isim tırnak işareti arasında 5 karakter olarak gözükse de `Ahmet` isminin arkasına 15 boşluk yerleştirilerek bellekte saklanmaktadır.

ÖRNEK 2: Bir kütükteki bilgileri, maksimum 80 sütundan oluşan satır olarak okuyup, kütükteki bir kelimenin/sözcüğün varlığını araştıracak, kelimeyi bulduğunda hangi satır ve sütunda yer aldığını bildiren bir program yazınız.

Kütük ismi ve aranan kelime veya sözcüğü, en fazla 12 karakter uzunluğunda, değişkenler olarak seçiyoruz. `SATIR` ile 80 karakterden oluşan satırı bir alfa sayısal olarak okutuyoruz. Aranan kelime veya sözcükteki boşlukları çıkarmak için `LEN_TRIM` fonksiyonunu kullanarak gerçek uzunluğunu tespit ediyoruz. (Bu işlemi yapmazsak, kelimenin arkasındaki boşlukları olan alfa sayısalı arayacaktır.) Böylece kelimenin boşluklar içermeyen sözcüğünü arama amaçlı kullanıyoruz. `READ` deyimi ile `IOSTAT` durumu takip edilmektedir ki kütük sonuna ulaşıldığında hata vermesin. `SATIR` içinde geçen `kelime`'yi `INDEX` fonksiyonu ile tarıyoruz.

```
PROGRAM Ornek2
IMPLICIT NONE
CHARACTER (LEN=80) :: SATIR      ! Satır bilgisi
CHARACTER (LEN=12) :: kutuk_ismi ! Kütük ismi
CHARACTER (LEN=12) :: kelime     ! Kütükte aranan kelime
INTEGER :: i, n, durum, uzunluk, d_durum
PRINT*, 'Kütük ismini giriniz => '
READ '(A)', kutuk_ismi
PRINT*, 'Aranacak kelimeyi giriniz => '
READ '(A)', kelime
!   Mevcut kütüğü okumaya aç
DO
   OPEN(3,FILE=kutuk_ismi,STATUS='OLD',IOSTAT=d_durum)
   IF(d_durum==0) THEN
      EXIT
   ELSE
      PRINT*, 'Kütük klasörünüzde mevcut değil '
   END IF
END DO
uzunluk=LEN_TRIM(kelime)
PRINT*, kelime(1:uzunluk),' kelimesi '
n=0
DO
  READ (3,'(A)',IOSTAT=durum) SATIR ! Satırı oku
  IF(durum==-1) EXIT                ! Kütük sonuna ulaşıldı
  n=n+1                             ! Satır sayacı
  ! kelime, SATIR'da mevcut değil ise i=0 olacaktır…
  i=INDEX(SATIR, kelime(1:uzunluk))
  IF(i/= 0) PRINT*, '**', n,'satırda ve ',&
            i,'.ci sütunda mevcut'
END DO
PRINT*, 'Kütükte ',n,' satır tarandı... '
END PROGRAM Ornek2
```

Bu programı test etmek amacıyla `B.DAT` isimli kısa bir kütük oluşturalım. Bu örnek problem için kütüğün içeriği aşağıdaki şekilde düzenlenmiştir:

```
         1         2
12345678901234567890
 bir iki üç
 dört beş bir
 alt on
 altı bir  iki
 sekiz
 on sekiz
 bir
 beş bir
 onbir dört
 on iki
```

Program çalıştırıldığında aşağıdaki çıktı elde edilir:

```
Kütük ismini giriniz => B.DAT
Aranacak kelimeyi giriniz => bir
bir kelimesi
** 1 satırda ve  2 .ci sütunda mevcut
** 2 satırda ve  11 .ci sütunda mevcut
** 4 satırda ve  7 .ci sütunda mevcut
** 7 satırda ve  2 .ci sütunda mevcut
** 8 satırda ve  6 .ci sütunda mevcut
Kütükte  10  satır tarandı...
```

8.5 DEĞİŞKEN-UZUNLUKLU ALFA SAYISAL FONKSİYONLAR

Alt programlara argüman olarak geçirilen alfa sayısalların `CHARACTER(LEN=*)` ile sabit uzunlukta tanımlandığını biliyoruz. Buna ilave olarak, Fortran herhangi bir uzunlukta alfa sayısal oluşturmamıza (*otomatik uzunluklu alfa sayısal fonksiyonu*) olanak sağlar. Örneğin, aşağıdaki modülde `alfa` fonksiyonu `alfabe`'nin ilk `n` karakterine eşit olmaktadır.

```
MODULE Degisken_uzunluk
CONTAINS
  FUNCTION alfa(n)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: n     ! Arzulanan çıktı uzunluğu
  CHARACTER(LEN=n) :: alfa     ! Çıktı alfa sayısalı
  CHARACTER(LEN=26)::alfabe= &
           'abcdefghijklmnopqrstuvwxyz'
     alfa = alfabe(1:n)
  END FUNCTION alfa
END MODULE Degisken_uzunluk
```



Otomatik-uzunluklu karakter fonksiyonları bir bariz arayüz ile tanımlanmalıdır. Aksi takdirde derleyici çıktı alfa sayısalının detaylarını bilmez!

8.6 SIRALAMA ALGORİTMALARI

8.6.1 SEÇİMLE SIRALAMA

Seçimle sıralama algoritması en kullanışlı algoritmalardan biridir. Verilen listedeki `N` elemanın içinde en küçüğü tespit edip, bunu listenin başına getir, sonra kalan `N-1` eleman içinden en küçüğü tespit ederek listenin ikinci sırasına koyar ve işleme bu şekilde devam edilir. Böylece `N` eleman küçükten büyüğe doğru sıralanmış olur.

ÖRNEK 3: En fazla 1000 kişinin isminin yer alabileceği bir kütükten, isimleri okuyarak alfabetik sıraya koyan bir program yazınız.

Programı aşağıdaki şekilde yazmak mümkündür. `UST` kalan listedeki ilk elemanın adresidir. Eğer bir andaki isim o andaki minimumdan, yani `MIN` den, küçük ise `MIN`'i yeniden `Gecici` ile yer değiştirerek bulmaktayız. Sonra minimum listenin başına yerleştirilir.

```
PROGRAM Ornek3
 ! Seçimle Sıralama Algoritması
 CHARACTER(LEN=20), DIMENSION(1000)::isim(1000)
 CHARACTER(LEN=20) :: Min, Gecici
 INTEGER :: i, n, UST, durum
 OPEN(5,FILE='Veri.Dat')
 READ(5,'(A10)',IOSTAT=durum)(isim(i),i=1,1000)
 IF(durum/=-1) N=I-1       ! N kişinin ismi okundu
 DO UST=1,n
      Min=isim(UST)
      DO  i=UST+l,n
           IF(isim(i)<Min) THEN
                Gecici = isim(i)
                isim(i)= Min
                Min    = Gecici
           ENDIF
      END DO
      isim(ust)=Min
  END DO
  WRITE(6,*) (isim(i),i=1,N)
  END PROGRAM Ornek3
```

8.6.2 KÖPÜK SIRALAMA (BUBBLE SORT)

Bir başka basit algoritmada *köpük sıralama* olarak bilinir ve liste kısmen sıralanır. Sadece alfa sayısallar için değil, sayısal değerlerin sıralanmasında da kullanılabilen bu algoritmanın ardında yatan düşünce şu örnek ile açıklanabilir. Elimizdeki sayısal liste sırasıyla 17, 11, 6, 2, 99 ve 8 olsun.

1. İlk iki elemanı (17 ve 11) kıyasla ve sıralamak istediğiniz düzene uymuyorsa yer değiştir. Bu durumda 17, 11 den büyük olduğu için sıralarını değiştirirsek yeni sıralama 11, 17, 6, 2, 99 ve 8 olur.

2. Bir sonraki çifte, yani 17 ve 6, aynı kıyaslamayı uygula ve gerekirse yerlerini değiştir. Bu durumda yeni sıralama 11, 6, 17, 2, 99, 8 olur.

3. Bu işleme listedeki bütün değerleri tarayarak devam et ve yer değiştirme işlemlerini gerektikçe yap. Kısacası sıralamalar her bir aşama için şu şekilde gelişir: 11, 6, 2, 17, 99, 8 sonra 11, 6, 2, 17, 99, 8 ve en son 11, 6, 2, 17, 8, 99.

4. Bu son listeye ikinci kez aynı işlemler dizisini uyguladığımızda sıralama düzeni 6, 2, 11, 8, 17, 99 olur.

ÖRNEK 4: Köpük sıralama algoritmasını uygulayarak 100 adet tamsayıyı küçükten büyüğe doğru sıralayan bir FORTRAN programı yazınız.

```
PROGRAM Ornek4
IMPLICIT NONE
! Köpük Sıralama Algoritması
LOGICAL :: CEVAP
INTEGER, DIMENSION(100) :: A
INTEGER :: i, n, j, GECICI, durum
OPEN(5,FILE='A.DAT',STATUS='UNKNOWN')
READ(5,*,IOSTAT=durum) (A(I),i=1,100)
IF(durum/=-1) n=i-1  ! n adet tamsayı okundu
PRINT*, n, 'adet tamsayı okundu'
DO i=n,2,-1
    CEVAP=.FALSE.
    DO j=1,i-1
        IF(A(j)>A(j+1)) THEN
             GECICI= A(j)
             A(j)  = A(j+1)
             A(j+1)= GECICI
             CEVAP=.TRUE.
        ENDIF
    END DO
    IF(.NOT.CEVAP) EXIT  ! cevap=.TRUE. olunca çık
END DO
WRITE(*,'(i5)') (A(i),i=1,n)
END PROGRAM Ornek4
```

CEVAP mantıksal değişkeni kullanılarak yer değiştirme yapılıp yapılmadığı kontrol ediliyor. Elemanlar arasında yer değiştirme yapıldığında CEVAP doğru değerini alıyor; yoksa baştaki yanlış değerini koruyor. Yer değiştirme yapılmadığında artık sıralama tamamlanmış demektir ki bu durumda döngüden çıkış sağlanıyor.

8.7 DAHİLİ KÜTÜKLER

Şu ana kadar sayısal ve alfa sayısal verilerin girdi/çıktı olarak programa ve programdan okutulmasını öğrendik; ancak sayısal değerleri alfa sayısala veya alfa sayısalı sayısal verilere dönüştürme konusundan bahsetmedik. Fortran bu tür dönüştürme işlemlerini yapmak için, *dahili kütük*, adı verilen özel bir mekanizmaya sahiptir.

Dahili kütükler Fortran I/O sisteminin, kütüklere yazılma ve okunmasından ziyade, bellekte dahili işlem gören özel bir uzantısıdır. Harici kütüklere kaydedilebilen her tür veri dahili kütüğe de kaydedilebilir. `READ` ve `WRITE` deyimlerinin genel şekli

```
READ (tampon,format) argüman_1, argüman_2,...
WRITE(tampon,format) argüman_1, argüman_2,...
```

Olarak verilir. Burada `tampon` alfa sayısal girdi tampon belleğidir. `Format` okuma ve yazma formatıdır.

Dahili kütük kullanımı sıklıkla alfa sayısal veriyi sayısala, yada sayısal veriyi alfa sayısala dönüştürmede kullanılır. Örneğin,

```
CHARACTER(LEN=8) :: veri='123.4567'
REAL :: deger
READ(veri,*) deger
```

Programında `veri` alfa sayısal bir sabittir (sayısal değil), ancak READ deyimi ile veri sayısal olarak `deger=123.4567`'e dönüştürülmüştür.

```
CHARACTER(LEN=132) :: tampon
REAL :: a=12.34, b=45.67
INTEGER :: i=11, j=24
WRITE(tampon,100) i, j, a, b
100 FORMAT(2(2x,i5),2(2x,f7.3))
PRINT '(A)', tampon
```

Programı ile `i`, `j`, `a` ve `b` değişkenleri tampon ile temsil edilen dahili kütüğe 100 numaralı format uyarınca alfa sayısal olarak kaydedilmiştir. Bu kaydı görmek için `PRINT` deyimini kullandığımızda çıktı aşağıdaki gibi olacaktır:

```
         1         2         3    3
12345678901234567890123456789012345
     11     24   12.340   45.670
```

ÖRNEK 5: Uzunluğu 20 karakter olan bir alfa sayısalı büyük harflerden oluşan bir alfa sayısala dönüştüren `BHarf` isimli bir alt programı yazınız.

ASCII tablosu incelendiğinde *büyük harfler* 65'den, *küçük harfler* ise 97'den itiraben kodlanmıştır; yani bir büyük harf ile küçük harf arasındaki fark 32'dir.

`Bharf` alt programında `alfaS` alfa sayısalı birinci karakterden son karaktere kadar karakter bazında küçük harfe karşılık gelip gelmediği araştırılıyor. Eğer karakter küçük harf ise, büyük harfe dönüştürmek için küçük harfin ASCII kodundan 32 çıkarılarak büyük harfin ASCII kodu bulunuyor. Son olarak da bu ASCII koduna karşılık gelen karakter ACHAR fonksiyonu ile bulunarak, alfa sayısaldaki yerine yazılıyor.

```
PROGRAM Ornek5
IMPLICIT NONE
CHARACTER(LEN=20) alfaS   ! alfa sayısal değişken
WRITE (*,*) ' En fazla 20 karakterden oluşan bir', &
   ' alfa sayısal giriniz'
READ (*,'(A20)') alfaS
CALL BHarf(alfaS)
WRITE (*,*) ' Dönüştürülen alfa sayısal ', alfaS
END PROGRAM Ornek5

SUBROUTINE BHarf(alfaS)
IMPLICIT NONE
CHARACTER(LEN=*), INTENT(INOUT) :: alfaS
INTEGER :: i
INTEGER :: uzunluk
uzunluk = LEN(alfaS)
DO i=1,uzunluk
   IF(LGE(alfaS(i:i),'a').AND.LLE(alfaS(i:i),'z')) THEN
      alfaS(i:i)=ACHAR(IACHAR(alfaS(i:i))-32)
   END IF
END DO
END SUBROUTINE BHarf
```

ALIŞTIRMALAR

8.1 Aşağıdaki program parçasını kullanarak çıktılarını bulunuz.

```
CHARACTER(LEN=6) X,Y,
CHARACTER(LEN=6), DIMENSION( 0:4) :: A
CHARACTER(LEN=6), DIMENSION(-1:2) :: B
CHARACTER(LEN=1) :: C, E
E='E'
C=E
X=' '//C
Y='NUMARA'
A(0)  = Y(2:)
BB(1) = Y(3:5)
BB(-1)= X//A(0)
A(1)  = BB(-1)
BB(2)(3:)= A(1)//E//X
A(4)  = X//A(2)//BB(1)
66 FORMAT(1X,A)

a) WRITE(*,66) C
b) WRITE(6,66) Y
c) WRITE(6,66) BB(-l)
d) WRITE(6,66) BB(0)//X
e) WRITE(6,66) C//E//Y
f) WRITE(6,66)(A(i),i=0,4)
g) WRITE(6,66)(BB(i),i=-1,2)
```

8.2 Aşağıdaki program parçasının çıktısını bulunuz.

```
PROGRAM p2
IMPLICIT NONE
CHARACTER(LEN=132) :: tmp
REAL :: a, b
a=REAL(1700/2400)
b=REAL(1700)/2400
WRITE(tmp,100) i, j, a, b
100 FORMAT(T11,i10,T31,i10,T51,F10.4,T28,F10.4)
END PROGRAM p2
```

8.3 En son karakter olarak nokta (`.`) içeren bir alfa sayısal sabiti okuyup, bunu tersinden yazan bir program hazırlayınız.

8.4 Bir Fortran kaynak programını okuyup içinden açıklama deyimlerini (ünlem işaretinin bulunduğu satır var mı? Varsa ünlem işaretinin olduğu kısmın çıkarılması) çıkardıktan sonra başka bir isimle kaydeden bir program yazınız.

8.5 `K` satır içeren bir paragraftaki kelimelerin sayısını hesaplayan bir program yazınız. *NOT:* Program paragrafın bulunduğu kütüğü girdi olarak okumalıdır; ayrıca her bir satırın en fazla 80 karakter uzunluğunda olduğunu, nokta, virgül gibi imla işaretlerinden sonra bir boş karakter (`' '`) konulduğu ve kelimelerin – ayracı ile hecelenmediğini kabul ediniz.

8.6 Bir Fortran kaynak programındaki `FUNCTION` ve `SUBROUTINE`'leri ayrı ayrı sayan ve bildiren bir program yazınız.

8.7 Herhangi bir metin kütüğünü okuyarak, metindeki "veya" kelimelerini "ve" ile değiştiren ve yapılan değişikliklerin sayısını bildiren bir program yazınız.

8.8 Girilen bir `INTEGER` sayıya son basamağından başlayarak her üç basamakta bir virgül ilave eden bir `CHARACTER FUNCTION` yazınız. Örneğin `12345678` sayısı `12,345,678` şeklinde yazması isteniyor.

8.9 Aşağıdaki programların çıktılarını bulunuz.

(a)
```
PROGRAM program_a
IMPLICIT NONE
CHARACTER(LEN=6) :: AD          !  B.dat kütüğü verileri
OPEN(4,FILE='B.DAT')           ! 'ALİ   '
DO WHILE(EOL)                  ! 'BİLAL '
  READ(4,10) AD                ! 'MEHMET'
 IF(A>'D') THEN                ! 'CEMİL '
    PRINT*, AD                 ! 'SEMİH '
 ELSE                          ! 'ZEKİ  '
    PRINT*, '  ',AD
  ENDIF
END DO
10 FORMAT(A6)
END PROGRAM program_a
```

(b)
```
PROGRAM Program_b
IMPLICIT NONE
CHARACTER(80) :: AlfaS, Ara
INTEGER :: IKON, i, Uzunluk=80
IKON=0
AlfaS='Bir varmış bir yokmuş. Evvel zaman içinde '
DO I=1,Uzunluk
IF(AlfaS(I:I) /= ' ') THEN
     IKON=IKON+1
```

```
     Ara(IKON:IKON)=AlfaS(I:I)
  ENDIF
  END DO
  PRINT*,AlfaS
  PRINT*,Ara
  END PROGRAM Program_b
```

8.10 Türkçe'de kullanılan bütün harfleri kapsayacak şekilde girilen tek bir karakterin büyük harf olup olmadığını kontrol edecek ve değilse büyük harfe dönüştürecek `Buyuk_Harf` isimli bir `FUNCTION` alt programı yazınız.

8.11 Alıştırma (8.10) ile verilen sorunun küçük harfe dönüştürme işlemini yapan `Kucuk_Harf` isimli bir `FUNCTION` alt programı yazınız.

8.12 Müşterilerin sayısını (`Musteri_Adet`), müşteri isimlerini içeren indisli değişkeni (`isim`) ve müşteri listesinde ismi olan kişi (`kisi`) gibi üç girdi parametresi olan ve `LOGICAL FUNCTION VarMI` olarak adlandırılması istenen bir alt program yazınız. Bu fonksiyon müşteri ismi (`kisi`) indisli listede (`isim`) mevcutsa `TRUE`, değilse `FALSE` değerini alsın isteniyor. *NOT:* İsimlerin maksimum uzunluğunu 20 karakter kabul ediniz.

8.13 Bir alfa sayısal indisli değişken `M` kişinin adlarım içermektedir, isimler alfabetik sıraya konduğunda ilk ve son gelen isimleri geri getiren bir `SUBROUTINE` yazınız.

8.14 Bir işadamı portföyündeki hisse senetlerini her gün gözden geçirerek, bazılarını satmakta ve ümit vaat eden yeni senetler satın almaktadır. Bu iş adamına ait bir günlük işlemi aşağıda verilmiştir. Bu verilere dayanarak aldığı ve sattığı hisse senetlerinin bedellerini hesaplayan bir program yazınız

Kod	*Hisse Senedi*	*Emir*	*Hisse* (lot)	*Fiyatı* (TL/lot)
004	Afyon Çimento	*AL*	800	35,00
006	Akbank	*AL*	500	11,00
016	Arçelik	*AL*	900	11,20
028	Brisa	*SAT*	300	7,20
035	Çimentaş	*SAT*	250	24,50
048	Döktaş	*AL*	420	12,10
065	KoçBank	*SAT*	600	1,55
085	İş Bankası (C)	*AL*	660	7,20
110	Migros	*SAT*	200	85,00
135	Ziraat Bank	*AL*	950	13,76
158	Tüpraş	*AL*	300	16,80
165	Vestel	*SAT*	100	3,10
169	Yapı Kredi Bank.	*AL*	440	3,30

BÖLÜM 9
VERİMLİ PROGRAM YAZMA TEKNİKLERİ

9.1 PROGRAMLAMA STİLİ

Programlama stilini tanımlamak oldukça zordur; fakat her programcının zamanla kazandığı bir stili vardır. Programlama stilinin karmaşık doğası, bilgisayar bilimcilerinin programlamayı bir bilimden ziyade "sanat" olarak tanımlamalarına neden olmuştur. Bu kısımda iyi bir programlama stilinin nasıl olması gerektiği konusuna değineceğiz. Bir programcı olmaya başladığınızda zamanla kendi programlama stilinizi ve kurallarınızı belirlemeye başlarsınız.

Stil insandan insana göre değişir. Programcılar olarak kullanılan derleyiciler, işletim sistemleri, donanım yapısı ve programlama ortamlarının farklılıklarından dolayı nadiren ortak geçmişi paylaşırız. Bu farklılıklar herkese uyumlu ve evrensel bir takım programlama kurallarının belirlenmesini imkansızlaştır-maktadır. Dolayısıyla, programlama stiline her bir programın belirli kullanımı ve çalıştırıldığı bilgisayar ve özellikleri bakımından fonksiyonel olarak yaklaşılmalıdır.

Doğru olarak yazılan bir program arzu edilen sonuçları üretir. Açıkça, bir "iyi" program aynı zamanda doğru olmalıdır. Genellikle belirli bir problemi çözen birçok doğru program yazılabilir; bunlardan bir kısmı diğerlerinden daha hızlı, kimisi daha hassas işlem yapan, bazıları daha az bellek ve yedek bellek gerektiren ve kimisi de çok güzel bir şekilde yapılandırılmış ve programa ilave yapılmak istendiğinde kolaylıkla yapılabilir durumda olabilmektedir, ideal bir program, bu bahsedilen özelliklerin hepsini içermelidir. Buna rağmen, gerçek dünyaya döndüğümüzde ideal programlara nadiren rastlıyoruz; çünkü programı iyi yapan etkenler bir birinden bağımsız değildir. Hesaplama hassasiyetinin artırılması genellikle bellek gereksiniminde bir artışa neden olacaktır. Bunun gibi bir programlama stratejisi yaparken programlamanın bellek gereksinimi gibi bir diğer parametresinin verimliliğini bozmak mümkündür.

Bu durumda amaç ne olmalıdır? Amaç mümkün olan en iyi programı verilen sürede, mevcut olanakları kullanarak hazırlamak olmalıdır. Program doğru olmalı, mümkün mertebe yeterli açıklamalar içermeli, bellek kullanımında tutumlu, hesaplama verimliliği olan, modüler ve son olarak da derleyiciden bağımsız (her derleyicide çalıştırılabilen) niteliklere sahip olmalıdır.

9.1.1 İYİ PROGRAMLAMA STİLİNİN TEMEL AKSİYONLARI

İyi bir programlama stili kazanmak için bazı tutumlar edinmelisiniz. Bunlar:

İyi bir tasarım ile çalışın: Bir programı dikkatli bir planlama olmaksızın çözümlemeye (hata ayıklama) ve değiştirmeye çalıştığınızda işleminiz bir işkenceye dönüşecektir. Bilgisayarın başına geçerek, program yazma arzusu herkeste fazladır; fakat bu arzuyu yenmek gerekir. Program kodlanmadan önce mutlaka iyi bir şekilde tasarlanmalıdır. Hata tespitinin kolay olması için organizasyonun mantıklı ve anlamlı olması gerekir.

Açık olun: Programınızı okuyacak kişi sadece siz olmayacaksınız. Belki de aradan geçen bir kaç ay sonra kendi programınıza bir göz attığınızda ne yaptığınızı unutmuş olduğunuzu fark edebileceksiniz. Uygun değişken ve alt program veya fonksiyon isminin seçimi ve programın içine yeterli açıklamaların serpiştirilmesi için çok az bir zaman sarfetmek yeterlidir. Bir programa iyi bir dokümantasyon sağlanmadığı sürece program tamamlanmış sayılmaz.

Değiştir ve yeniden yaz: Bir programı yazdıktan sonra gözden geçiriniz. Çalıştırdığınızda doğru sonuç vereceğine inanıyor musunuz? Program anlaşılabilir midir? Programı daha verimli çalışacak bir hale getirebilir misiniz? Bu soruların ışığı altında, eğer programınızın önemli derecede geliştirmeye açık olduğuna inanıyorsanız (ki program her zaman daha da geliştirilebilir), bu durumda gerekli değişikliklerin yapılmasından kaçılmaması gerekir.

Gereksiz işlemlerden kaçının: Bir deyim gereksiz hesaplamalar, bir alt program gereksiz deyimler, değişkenler ve argümanlar içermemelidir. Fazlalıklar programda karmaşaya neden olur.

Standardları her zaman tercih edin: Çok sayıdaki programlama deyimlerinden bazıları nadiren kullanılır; bu nedenle, bunlara fazla dikkat etmeyiz. Fakat çoğu kez programlama dilinin bu özellikleri en kısa ve en doğru sonucu verir. Arzulanan işlevi gören arşiv fonksiyonları veya alt programlar öncelikle tercih edilmelidir. Örneğin, bir gurup sayının içinden en büyüğünü veya en küçüğünü tespit eden bir alt program yazılabilir, oysa `AMAX1` ve `AMIN1` alt programları, `REAL` sayılar ve bu iş için tasarlanmış kütüphane fonksiyonlarıdır.

Özel diyalektlerden kaçının: Çeşitli makineler için hazırlanan FORTRAN derleyicileri makine özelliklerine ve donanımına bağlı olabilmektedir. Hazırlanan programın bu spesifik özelliklere dayandırılması (derleyici-bağımlı) arzusu karşısında, programlama basitliği ve verimliliği temin edebilirsiniz; fakat programınızı başka bir sistemde çalıştırmaya kalkıştığınızda problemlerle karşılaşırsınız. Amerika Ulusal Standardlar Enstitüsü (American National Standards Institute - ANSI) FORTRAN dilini ve standart deyimlerini tanımlamıştır. ANSI standartlarına uyan programlar bütün FORTRAN derleyicilerinde derlenebilirler.

9.1.2 DOKÜMANTASYON

Dokümantasyon bir programa eşlik eden ve programın nasıl çalıştığını açıklayan materyali tanımlar. Dokümantasyon, aslında, programın bir kısmı da (anlamlı değişken isimleri ve açıklamalar) olabildiği gibi bir kullanım kılavuzu şeklinde de olabilir.

Dokümantasyon miktarı, programın ömrüne, karmaşıklığına ve kullanım yaygınlığına bağlı olmalıdır. Her programın kendine has özellikleri olmasına rağmen, dokümantasyon ile ilgili bazı genel tavsiyeler aşağıda sıralanmıştır:

- Daima ihtiyacınız olduğuna inandığınızdan fazla dokümantasyon temin ediniz;
- Dokümantasyonu mümkün mertebe program kodu içinde yapınız;
- Programınızı açık ve anlaşılır kılınız;
- Temiz ve düzenli olunuz.

9.1.2.1 FORTRAN DEYİMLERİNİN TANIMLANMASI

Sabit-kaynak stilinde (fixed-source format) yazılan bir FORTRAN programı, bir satırının 73 ile 80 sütunları arası programı tanımlama, satır sırasını belirleme ve değişiklik yapılan satırları işaretleme olarak üç amaç için kullanılabilir. Genellikle 73-76 sütunlar belirli bir program veya alt programı isminin ilk dört harfi ile tanımlama, 77-80 sütunlar da numaralama amacıyla kullanılırlar. Bu prosedür satır numarası 10'arlı artışlarla bin satıra numaralamaya olanak sağlar. Numaralandırmanın l0'arlı şekilde artırımının nedeni, daha sonra program arasına yapılabilecek yeni ilavelere kolaylıkla numara verilmesi amaçlanır. Örneğin, indisli olarak verilen `n` adet sayının ortalamasını hesaplayan bir `FUNCTION` alt programı bahsedilen prosedüre uygun olarak aşağıdaki şekilde hazırlanabilir:

```
         1         2                             7         8
12345678901234567890 1 . . . . . . . . . . . .56789012345678900
-----------------------------------------------------------
      REAL FUNCTION Ortalama(n,x)                     ORTA0010
      INTEGER, INTENT(IN) :: n                        ORTA0020
      REAL, DIMENSION(n), INTENT(IN)::x               ORTA0030
      INTEGER :: i                                    ORTA0040
      REAL :: Ortalama, Topla                         ORTA0050
      Topla=0.0                                       ORTA0060
      DO i=1,n                                        ORTA0070
         Topla=Topla+x(i)                             ORTA0080
      END DO                                          ORTA0090
      Ortalama=Ortalama/REAL(n)                       ORTA0100
      END FUNCTION Ortalama                           ORTA0110
```

Serbest-kaynak stilinde (fixed-source format) hazırlanan bir FORTRAN programı yukarıdaki kısıtlamalara tabi değildir. Adından da anlaşılacağı üzere, program deyimleri bir satıra istendiği sütundan itibaren yazılabilir. Satır uzunluğu en fazla 132 karakterdir. Bir satıra sığmayan deyimler, satırın sonuna 'alt satırda devam ediyor' anlamına gelen & işareti konulur.

```
         1         2                             7
123456789012345678901 . . . . . . . . . . . .567890
PROGRAM Deneme
INTEGER :: m=40
INTEGER, DIMENSION(m) :: a
 a(1) = a(m) + 2.*a(m-1)+3.0*a(m-2)+ &
        4.*a(6) + 3.*a(5) + 2.*a(4)+ &
        3.*a(3) + 2.*a(2)
10 FORMAT(5(4x,f16.6))
20 FORMAT(2x,5E12.4)
END PROGRAM Deneme
```



Fortran 90/95 programı hazırlarken serbest-kaynak formunu kullanınız!

9.1.2.2 AÇIKLAMA SATIRLARININ İLAVE EDİLMESİ

En güçlü programlama araçlarından birisi de kaynak programlarda açıklama satırlarının kullanılabilmesidir. "Anlaşılır" ve "stratejik yerlere" yerleştirilen açıklamalar, programcının amacını ortaya koyar. Dikkatlice yazılan açıklamalar, programın faydasını ve gerektiğinde üzerinde değişiklik yapma olanağını kazandırır. Örneğin aşağıdaki açıklama satırlarını göz önüne alalım:

```
     . . .
!
!       n'in DEĞERİNİ 3 ARTIR
!
n=n+3
     . . .
!------------------------------------------------------
!       MATRİSİN MAKSİMUM BOYUTUNU 240'a AYARLA
!------------------------------------------------------
max_boyut=240
```

Birinci açıklama bloğu sadece `n`'in değerinin 3 artırıldığını belirtiyor; fakat ikinci blok matrisin boyutunun maksimum 240 olarak ayarlandığını belirtmektedir. Bu bakımdan ikinci blok daha anlamlı ve açıklayıcı iken, ilk blok sadece aritmetik işleminin yazı ile açıklanmasından başka bir işe yaramamaktadır.

Açıklamalar herhangi bir sütunda ünlem işareti (`!`) yerleştirilerek, o satırın işaretli sütundan sonraki sütunlarını tamamen açıklamaya ayrılır. Örneğin,

```
n=n+3              ! n'nin değerini 3 artır
Sicaklik=123.5     ! Sıcaklık birimi santigrat alındı
Kütle   = 5.       ! Ağırlık birimi Kg'dır
```

Bir diğer faydalı teknik de bazı önemli işlemlerin yapıldığı alt programlar ve modüllerin çerçeve içinde verilmesidir. Uzun programlarda çok sayıda alt program mevcut olabilir. Bu alt programlara ulaşmak isteyen programcı üzerinde çalışacağı kısmı aramasının kısa sürmesi için kullanılır. Örneğin,

```
!
!       BAŞLANGIÇ DEĞERLERİNİ ATA
!
!
!
!    ********************************
!    *                              *
!    *  MATRİSİN TERSİNİ HESAPLA    *
!    *                              *
!    ********************************
!
```

şeklinde program modülleri veya bloklarının başı ve/veya sonu belirginleştirilir.

Diğer taraftan, bir kitabı elinize aldığınızda, bu kitabın adı, yazarı, basım tarihi, kullanılan kaynaklar v.s gibi bilgileri içerdiğini görürsünüz. Benzer şekilde, her programın başında bu gibi bilgiler yer almalıdır:

```
!============================================================
!      PROGRAM — MAAŞ HESAP
! **   PROGRAMCI - NACİ E. ÖZTÜRK  **
! *
!    TARiH - 8 KASIM 2004
!
!    TANIM - BU PROGRAM SAĞLIK KURULUŞLARINDA ÇALIŞAN
!    PERSONELİN MAAŞ VE TAZMİNATLARINI HESAPLAYAN
!    BİR PROGRAMDIR.
!
!    UYARI!- MAKSİMUM 1500 KIŞININ MAAŞINI HESAPLAR.
!    PROGRAMDA SON YAPILAN DEĞİŞİKLİK TARİHİ 27/3/2005
!============================================================
```

Programda kullanılan değişkenler ve sabitler, yine ana ve alt programların başında veya hizasında (bu durumda her değişken bir satırda tip ile beraber tanımlanır), aşağıdaki gibi açıklamalar ile tanımlanması, programı ilaveler veya değişiklikler yapmak için inceleyen birisi için, programı takip edebilmesine olanağı sağlar.

```
!
!    MIN   = Minimum İndis Değeri
!    MAX   = Maksimum İndis Değeri
!    M_BOY = Matrisin Boyutu
!    A_mat = (M_BOY X M_BOY) Boyutlu Matris
!    SSAPM = Standard Sapma
!    XORT  = X'lerin Ortalaması
!
```

veya

```
     INTEGER :: i              ! İndis değişkeni
     REAL    :: toplam         ! Sayıların toplamı
     REAL    :: sabit=12.35    ! Katsayı çarpanı
     REAL    :: Ortalama       ! n sayının aritmetik ortalaması
```

Bazen programlarda, özellikle tamsayılarla hesaplarda, bir eklenmesi veya çıkarılması durumlarında yeni değişkenler tanımlanmaktadır. Örneğin,

```
     I1=I+1
     J1=J-1
     L3=L+3
```

Eşitliğin solundaki yeni tamsayı değişkenlerinin isimlerini programcının kolaylıkla kendi hafızasında tutabileceği isimler vermesini sağlamak aşağıdaki gibi mümkün olabilmektedir:

```
     I_Arti_1=I+1
     J_Eksi_1=J-1
     L_Arti_3=L+3
```

9.2 HESAPLAMA SÜRESİNİN AZALTILMASI

Bir programı hazırlarken, hazırlanan programın sadece arzulanan işlemleri yapmasını istemek ve beklemek yeterli değildir; programcı, program işlemlerini en hızlı şekilde yapacak tarzda yazmalıdır. Programın çalışmasını hızlandırma problemi, çalışma süresi kısa olan ya da az sayıda aritmetik veya mantıksal işlem gerektiren programlar için o kadar önemli değildir; ancak çalıştırıldığında saatler veya günlerce süren programlar için çalıştırma süresini (execution time) minimum sayıda aritmetik ve/veya mantıksal işlem yapmak ile azaltmak mümkündür. Maalesef, bu bazen bellek ve ikinci bellek gereksinimini, ayrıca hiç şüphesiz programlama çabalarını artırır.

Günümüzde bilgisayarlar çok hızlı bir şekilde gelişmekte ve her yeni model bir öncekinden daha hızlı tasarlanmaktadır. Bir bilgisayarın fiyatı hızı ile doğru orantılı bir şekilde arttığı gözlenmektedir. Herhangi bir program çalıştırıldığı bir bilgisayarda 20 dakika sürüyorsa, bir başka bilgisayarda 3 dakika sürebilir; ama daha hızlı bilgisayarın fiyatı da daha pahalı olacağından, daima en hızlı bilgisayarı satın almak da her zaman akılcı bir çözüm değildir. Programın çalışma süresini kısaltmak, hesaplama süresini büyük ölçüde azaltmak, bazı akıllı programlama teknikleri ile mümkündür. Bu kısımda da programlamada zaman tasarrufu sağlayacak ve program yazmayı kolaylaştıracak bazı tekniklere değineceğiz.

Bir programın hesaplama süresini azaltmak için harcanması gerekli çabanın saptanmasında aşağıdaki faktörler göz önüne alınır:

- programın kaç kez çalıştırılacağı,
- hesaplama süresinde sağlanacak tasarruf miktarı,
- program geliştirmeye ayrılan süre,
- programın karmaşıklığı,
- bilgisayar zamanının maliyeti,
- programcı zamanının maliyeti.

Sadece bir kez kullanılacak bir programda bu faktörler fazla önemli olmayabilir; fakat çok sık kullanılan ve uzun süre çalışan programlarda oldukça önem kazanırlar.

Bilgisayarlar çeşitli hızlarda üretildiklerinden, bir sene önce üretilen bir bilgisayar çok çabuk demode olabilmekte ve hız bakımından yeni modellere oranla yavaş kalabilmektedir. Dolayısıyla, bir aritmetik işleminin bilgisayarda hesaplanma süresi farklılıklar göstermektedir. Aritmetiksel işlemler mikrosaniye (μs) mertebesinde yapılmaktadır. Bu işlemlerin bilgisayarda nispi olarak yaklaşık hesap süreleri aşağıdaki tabloda sunulmaktadır.

İşlem	**Gösterim**	**süre (μs)**
Toplama	`X+Y`	0.4
Çıkarma	`X-Y`	0.4
Çarpma	`X*Y`	1
Bölme	`X/Y`	3
Üst Alma	`x**y`	12
Karekök	`SQRT(X)`	23
	`X**0.5`	94

Bu rakamlar hesaplanan bilgisayarda çarpma işlemi "bir" olacak şekilde normalize edilmiştir. Başka bilgisayarlarda çarpma işlemi 1 μs olmayabilir; fakat diğer işlemler ile aradaki oran yaklaşık olarak yukarıdaki tabloda verilen değerler civarındadır. Bu da bize programda sağlanacak zaman tasarrufu hakkında nispi bir bilgi verir.

Çalıştırma sürelerinin detaylı olarak tespiti için bilgisayar üreticisinin kılavuzuna başvurulmalıdır; fakat aşağıdaki genel kurallar uygulanır: `INTEGER` aritmetik `REAL` aritmetikten, `REAL` aritmetik de `DOUBLE PRECISION` aritmetikten, toplama ve çıkarma çarpmadan, çarpma işlemi bölmeden, bölme işlemi üst almadan v.s daha hızlıdır.

9.2.1 ARİTMETİK İŞLEM SAYISININ AZALTILMASI

En sık karşılaşılan hesaplama verimsizliğinin sebeplerinden biri de gereksiz ifadelerin defalarca hesaplanmasıdır. Örneğin, ikinci dereceden denklemin kökleri

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

olarak verilmektedir. Fazla tecrübeli olmayan bir programcı, köklerin hesabını aşağıda verilen aritmetik atama deyimleri şeklinde programlar.

```
xl=( -b + SQRT(b**2-4.*a*c) )/(2.0*a)
x2=( -b - SQRT(b**2-4.*a*c) )/(2.0*a)
```

Buradaki deyimler fazla sayıda tekrarlayan (gereksiz) aritmetik işlemler içermektedir. Örneğin, **`b**2-4.*a*c`** ifadesi sadece bir kere hesaplanabilir; böylece karekök içindeki işlemin hesabı iki kez tekrarlanmaz:

```
Delta=b**2-4.0*a*c
xl=( -b + SQRT(Delta) )/(2.0*a)
x2=( -b - SQRT(Delta) )/(2.0*a)
```

Eğer programda `Delta` değerine herhangi bir nedenle ihtiyaç yoksa karekök işlemini ve payda değeri olan **`2.0*a`**'yı iki kere hesaplamaktan kurtarabiliriz. Böylece

```
Payda=2.0*a
Kok_Delta=SQRT(b**2-4.0*a*c)
xl=( -b + Kok_Delta )/Payda
x2=( -b - Kok_Delta )/Payda
```

halini alır. Daha *kurnazca* bir yaklaşımla yapılabilecek bir başka işlem de, paydayı **`2.*a`** çarpımı yerine **`a+a`** toplamı, **`b**2`**'yi de **`b*b`** çarpımı olarak ifade etmektir. Bu durumda program

```
Payda=A+A
Kok_Delta=SQRT(b*b-4.0*a*c)
xl=( -b + Kok_Delta )/Payda
x2=( -b - Kok_Delta )/Payda
```

şeklini alır. Bu son durum başlangıçtaki ifade gurubundan kesinlikle daha verimli bir aritmetik deyim gurubu oluşturmaktadır. Yalnız burada verimliliği artırmamıza rağmen algoritmanın açıklığında bir azalmaya neden olduğumuzu gözden kaçırmamak gerekir.

Bütün bu işlemlerle hesaplama süresi kısaltılmaktadır. Bu kısalma oranına ve yapılabilecek başka kısaltma tekniklerine bir göz atalım:

Bir programda x^2 , x^3 gibi kuvvetlerin hesabı gerekiyorsa, bu işlemler aşağıdaki şekilde yapılabilir.

```
(a)                    (b)
REAL :: X,..           REAL :: X,..
READ*, X               READ*, X
X2=X*X                 X2=X**2
X3=X2*X                X3=X**3
X4=X3*X                X4=X**4
```

Burada (a) şıkkında verilen programda üst alma işlemleri üç çarpma işlemi ile 3μs de yapılmaktayken (b) ile verilen programda üç üs işleminden dolayı 36 μs 'de yapılmaktadır.

Örneğin, $y = a + bx + cx^2 + dx^3 + ex^4$ şeklinde verilen bir polinomun Fortran dilinde kodlanmasını ele alalım. Bir çok programcı bunu otomatik olarak

```
Y = A + B*X + C*X**2 + D*X**3 + E*X**4
```

şeklinde kodlar. Burada yapılan işlemler 3 üst alma (3×12=36 μs), 4 çarpma (4×1=4 μs) ve 4 toplama işlemi (4×0.4=1.6 μs) yapılarak, 41.6 μs sürecekir. Oysa bu formülü aşağıdaki gibi

```
Y=A+X*(B+X*(C+X*(D+E*X)))
```

şeklinde parantezlere ayrılmış ($y = a + x(b + x(c + x(d + ex)))$) olarak kodlayarak üst alma işleminden de kaçınabiliriz. Toplam işlem süresi (4 çarpma + 4 toplama işlemi) 5.6 μs olmaktadır.

Benzer şekilde, aşağıdaki matematiksel formülün programlanmasını ele alalım.

$$Z = \left[\frac{a+b}{\sqrt{a^2-b^2}} + \frac{\sqrt{a^2-b^2}}{a+3b} \right] \div \left[\sqrt{a^2-b^2} + a + b \right]$$

Bu formülü

```
Z=((A+B)/(A**2-B**2)**.5+(A**2-B**2)**.5/(A+3.*B))/ &
  +((A**2-B**2)**.5+A+B)
```

şeklinde programladığımızda 8 toplama ve çıkarma, l çarpma, 6 üst alma, 3 bölme ve 3 karekök hesabı (üst şeklinde) yapılmaktadır. İşlem süresi 367.2 μs bulunur. Oysa

```
C=SQRT(A*A-B*B)          ! 3 kez tekrarlanan terim
D=A+B                    ! 3 kez tekrarlanan terim
Z=(D/C+C/(C+2.*B))/(C+D)
```

şeklinde programlayınca hem hata yapma olasılığı azalmakta hem de işlem süresi 37 μs'ye düşmektedir. Diğer taraftan kullanılan değişken sayısı artmakta bu kez programın kullandığı belleği artırmaktadır.

Az sayıda değişken kullanarak, bellekten yana tasarrufta bulunmak için, aşağıdaki ifadenin

FORTRAN olarak kodlanmasına şöyle bir örnek de verilebilir.

$$y = \frac{abx}{1+c} + \frac{rP}{(1+r)^n - 1}$$

Cebirsel işlemi hem parçalayarak, hem sayısal değerlerin hafızaya yerleştirilmesi kuralından yararlanarak, hem daha az sayıda "gerçek" değişken kullanarak şu şekilde kodlayabiliriz.

```
Y1=1.+C          ! Y1 <= 1+c
Y1=A*B*X/Y1      ! Y1 <= abx/Y1       yani  Y1 <= abx/(1+c)
Y2=1.+R          ! Y2 <= 1+r
Y2=Y2**N-1.      ! Y2 <= Y2^n - 1     yani  Y2 <= (1+r)^n - 1
Y2=Y2/R          ! Y2 <= Y2/r         yani  Y2 <= [(1+r)^n - 1]/r
Y=Y1+P/Y2
```

Programlarda, sık sık aşağıdaki örneklere benzer şekilde, bir değişkeni veya aritmetik işlem gurubunu sabit bir sayıya bölüme işlemi uygularız:

```
X=(A+3.*Z**(1./4.))/3.0
A=C/2.0+X/5.0
```

Aritmetik deyimleri bu şekilde kodlamaktan kaçınıp, bölme işlemlerini çarpma işlemine çevirmekte yarar vardır; çünkü zamandan yaklaşık % 70 tasarruf sağlanmaktadır. Yani. yukarıdaki deyimler

```
X=0.3333333*(A+3.*Z**0.25)/
A=0.5*C+0.2*X
```

olarak yazılabilir. Dikkat edilecek olursa 0.3333333 sabitinin sadece 7 basamağı girilmiştir. Bunun nedeni de tek hassasiyetle çalışırken, hatırlanacağı üzere, bellek depolama kapasitesi yaklaşık 7 basamak olmasıdır.



Sabit bir sayıya bölme veya üs alma işlemlerinden "mümkün olduğu kadar" kaçınınız; aritmetik ifadelerinizi daha ziyade çarpma işlemi şeklinde yapınız. `x/2=0.5*X   x**3=x*x*x`

Mühendislik hesaplarında sabitler (π ve e sayısı gibi) sıkça kullanılır. Bunları, kullanıldıkları programların başında, aşağıdaki şekilde tanımlamak sabitin yedi, `REAL(KIND=8)` de ise 15-16 basamağa kadar doğru olarak hesaplanmasına sebep olur; ayrıca programda defalarca tanımlamaya gerek yoktur.

```
REAL :: pi, e
pi = 4.0*ATAN(1.0)
e  = EXP(1.0)
```

Burada $\tan^{-1}(1) = \pi/4$ ve $e = e^1$ olup arşiv fonksiyonlarının özelliklerinden faydalanılmıştır.



Çok uzun süre çalışan, milyarlarca aritmetik işlemin yapıldığı programlarda, aritmetik işlem sayısını azaltmak için azami gayret gösteriniz.

9.3 HESAPLAMA HASSASİYETİNİN ARTIRILMASI

Bilgisayarlar bile hata yaparlar. Bir bilgisayar her ne kadar iki kere ikiyi beş olarak hesaplamasa veya toplama işlemi yapacağı yerde çarpma işlemi yapmasa da, daha karmaşık hatalar yapabilmektedir. Bu hatalardan bilgisayara güvenememe ya da bilgisayarın her işlemini elle kontrol etmek gerekir gibi bir anlam çıkarılmamalıdır; sadece program yazarken hesaplarda hatalara yol açabilecek durumların farkında olunmalı ve bu tür hatalara olanak verilmemelidir.

Sonuçlardaki yanlışlık bilgisayarın sayıları kullanma ve bellekte saklama yönteminden kaynaklanır. Matematik bilgimiz `(l./3.)*3.` işleminin "`l`" olması gerektiğini söylerken, bilgisayar farklı cevap vermektedir. `1./3.` bölümü `0.3333333....` iken `3`'lerin sayısı sonsuz olup, bu değer bilgisayarda doğru bir şekilde depolanamamaktadır. Bu değer 3. ile çarpılınca sonuç l'den biraz küçük olur; yani sayısal değerler, gerçek değerlerine en yakın değere yuvarlanarak depolanır. Örneğin, π, *e* ve `1./6. = 0.166666..` gibi sayıların sadece 7 basamağa kadar olan kısımları hassas olarak bellekte depolanır.

Bazen çok küçük sayılar bellekte, sahip oldukları değerlerin biraz altında, depolanırlar; örneğin `0.1` ve `0.02` gibi sayılar `0.0999999` ve `0.019999` olarak saklanır çünkü ikilik sayı düzende 0.1'in değeri 0.0001100110011... dir, yani tekrarlı bir sayıdır. Bu sayılar ile aritmetik işlemlerin çok sayıda tekrarlanması, hesaplanması gereken değerin son basamaklarında belirgin sapmalara neden olur. Aradaki fark başlangıçta çok az olmasına ve ihmal edilebilir görünmesine rağmen, sapmanın milyonlarca işlemde tekrarlanması daha büyük hatalara sebep olacak ve sonucu geçersiz kılacaktır.

Bu tür hataları ve bazı durumlarda bu hatalardan sakınma yollarını açıklamak bakımından l sayısına 100 adet 0.01 ilave etmeyi deneyelim. Bu işlemi yapmak için aşağıdaki şekilde bir program hazırlanmış olsun:

```
PROGRAM Test
REAL :: A=1.0, DT=0.01
INTEGER :: n=101, i
DO i=2,n
     A = A + DT
     WRITE(*,10) A
END DO
10 FORMAT(1x,F9.6)
END PROGRAM Test
```

Bu program 100×0.01+1=2 sonucunu vermesi gerekirken, en son değer milyonda bir eksik hesaplanmıştır. Bu eksikliğin nasıl oluştuğunu program çıktısı olan ara hesaplar ile birlikte aşağıda verilmektedir:

Sonuçlar aşağıda özetlenmiştir:

```
1.010000    1.020000    1.030000    1.040000    1.050000
1.060000    1.070000    1.080000    1.090000    1.100000
1.110000    1.120000    1.130000    1.140000    1.150000
1.160000    1.170000    1.180000    1.190000    1.200000
1.210000    1.220000    1.230000    1.240000    1.250000
1.260000    1.270000    1.280000    1.290000    1.300000
1.310000    1.320000    1.330000    1.340000    1.350000
1.360000    1.370000    1.380000    1.390000    1.400000
1.410000    1.420000    1.430000    1.440000    1.450000
1.460000    1.470000    1.480000    1.490000    1.500000
1.510000    1.520000    1.529999    1.539999    1.549999
1.559999    1.569999    1.579999    1.589999    1.599999
1.609999    1.619999    1.629999    1.639999    1.649999
1.659999    1.669999    1.679999    1.689999    1.699999
1.709999    1.719999    1.729999    1.739999    1.749999
1.759999    1.769999    1.779999    1.789999    1.799999
1.809999    1.819999    1.829999    1.839999    1.849999
1.859999    1.869999    1.879999    1.889999    1.899999
1.909999    1.919999    1.929999    1.939999    1.949999
1.959999    1.969999    1.979999    1.989999    1.999999
```

(*NOT: Bu çıktılar aslında alt alta ve tek sütun halindedir; fazla yer kaplamaması için yan yana düzenlenmiştir*.)

Burada görüldüğü üzere, özellikle 1.52’den sonra 1.53 elde edilmesi gerekirken 1.5299999 hesaplanmıştır. Bu işlemi devam ettirirsek, yani 1 sayısına 100 adet değil de, 200, 400, 800, 100 adet 0.01 ilave ettiğimizde, son değer ne olur? Bu soruyu cevaplamak için verilen `Test` programında 101 değerini değiştirerek bulabiliriz.

Sonuçlar aşağıdaki tabloda özetlenmektedir:

N	*Gerçek Değer*	*Hesaplanan Değer*
101	2.000000	1.999999
201	3.000000	2.999998
401	5.000000	5.000020
801	9.000000	9.000112
1001	11.000000	11.000160

Görüldüğü üzere mutlak hata, yani hesaplanan değerin gerçek değerden mutlak değerce sapma miktarı, artan aritmetik işlem sayısı ile artmaktadır; diğer bir deyişle, sayının son basamaklarında bozulma, işlem sayısı arttıkça ondalık noktasına doğru ilerlemektedir. İşlem sayısı 1000 iken, mutlak hata yüz binde 16’dır. Bu işlemi milyonlarca kez tekrarlamış olsaydık ne olurdu? sorusunu cevaplamaya çalıştığımızda işlem sonucundan artık emin olmayacağınız aşikardır.

Bu problemin üstesinden gelmenin bir yolu mevcuttur. Bu da toplama işlemini ondalıktan kurtardıktan sonra bölme işlemi yapmak suretiyle ile başarılabilir. Yani 0.01 sayısından kurtulmak için bunu 100 ile çarpmak gerekir; bu durumda l sayısı da 100 ile çarpılır ve 100 sayısına l'ler ilave edildikten sonra 100'e bölünürse, arzu edilen sonuç elde edilir.

Bu mantığa göre hazırlanan program aşağıdaki şekilde hazırlanabilir:

```
PROGRAM Test_Yeni
IMPLICIT NONE
REAL    :: B=100., C=1.0, A=1.0
INTEGER :: i, n=101
DO i=2,n
   B = B + C
   A = 0.01 * B
   WRITE(6,10) A
END DO
10 FORMAT(1X,F9.6)
END PROGRAM Test_Yeni
```

Programın `n=101` için çıktısı aşağıda verilmektedir. Çıktıda hatalar önlenmiş olmaktadır. Bu program `n`'in hangi değeri için olursa olsun, çalıştırdığında daima gerçek değeri verecektir.

```
1.000000  1.010000  1.020000  1.030000  1.040000  1.050000
1.060000  1.070000  1.080000  1.090000  1.100000  1.110000
1.120000  1.130000  1.140000  1.150000  1.160000  1.170000
1.180000  1.190000  1.200000  1.210000  1.220000  1.230000
1.240000  1.250000  1.260000  1.270000  1.280000  1.290000
1.300000  1.310000  1.320000  1.330000  1.340000  1.350000
1.360000  1.370000  1.380000  1.390000  1.400000  1.410000
1.420000  1.430000  1.440000  1.450000  1.460000  1.470000
1.480000  1.490000  1.500000  1.510000  1.520000  1.530000
1.540000  1.550000  1.560000  1.570000  1.580000  1.590000
1.600000  1.610000  1.620000  1.630000  1.640000  1.650000
1.660000  1.670000  1.680000  1.690000  1.700000  1.710000
1.720000  1.730000  1.740000  1.750000  1.760000  1.770000
1.780000  1.790000  1.800000  1.810000  1.820000  1.830000
1.840000  1.850000  1.860000  1.870000  1.880000  1.890000
1.900000  1.910000  1.920000  1.930000  1.940000  1.950000
1.960000  1.970000  1.980000  1.990000  2.000000
```

Bilgisayarın yetenekleri ve karakteristiklerinden bahsederken (Bölüm 1 Kısım 1.2.6) 2'nin kuvvetleri olan sayıların bilgisayarda tam olarak temsil edildiğinden bahsetmiştik. Bunlar 2, 4, 8, ... ve 0.5, 0.25, 0.125, ... gibi tam ve ondalıklı sayıları (ikinin negatif tam kuvvetleri) içermektedir.

Bu problemlerin üstesinden gelmenin bir başka yolu da programdaki sayıları (veya programı) `DOUBLE PRECISION/REAL(8)` olarak tanımlamaktır. Ancak bu sorunu tam olarak çözmez. Bu kısımda bahsedilen hata türlerinin alt ve üst sınırlarını değiştirir. Diğer taraftan aritmetik işlemlerin sebep olduğu yuvarlama hatalarına neredeyse kesin çözüm getirmektedir. Bunun nedeni de çift hassasiyetli sayıların 15-16 basamağının bellekte korunması ve aritmetiklerin bu basamaklara kadar hassas olarak yapılmasından kaynaklanmaktadır.

Çift hassasiyetli programlamaya örnek olarak, aşağıdaki programı ve sonuçlarını inceleyelim.

Bu program parçası tek hassasiyetli (SINGLE PRECISION) olarak yazılmıştır.

```
REAL :: x,pi
x  = 22.0/7.0
pi = 4.*ATAN(1.0)
WRITE(6,10) x, pi
10 FORMAT(2(1X,F18.15))
```

Bu programın çıktısı

```
3.142857000000000    3.141593000000000
```

olarak elde edilir; çünkü ilk altı basamaktan sonrası için atama yapılamadığından, bu basamakların değerleri sıfırdır. Oysa programı `DOUBLE PRECISION`'a dönüştürmek için `REAL` deyimini `DOUBLE PRECISION` ve `ATAN` fonksiyonunu da `DATAN` ile değiştirmemiz yeterli olacaktır. Bu arada sabitlerin de çift hassasiyetli olarak saklanması için ondalık noktasından sonra `D0` ilavesini yapmak gerekir. Bu durumda

```
REAL(KIND=8) :: x, pi
x  = 22.D0/7.D0
pi = 4.D0*DATAN(1.D0)
WRITE(6,10) x, pi
10 FORMAT(2(1X,F18.15))
```

programının çıktısı

```
3.142857142857143    3.141592653589793
```

olarak elde edilir. Bellekte depolanan basamak sayısı aynen çıktıda gözükür.

9.4 PROGRAMLAMADA BAĞIMSIZLIK, GENELLİK VE BÜTÜNLÜK

Bağımsızlık: Bir programın bağımsızlığı, farklı derleyici ve işletim sistemlerinde veya bilgisayarda ne kadar kolay çalıştırılabildiğinin bir ölçüsüdür. Bir program ne kadar bağımsız ise bu programı yeni bir bilgisayar ortamına aktarmak, bu ortamda (PC, Network, Çalışma istasyonu, Main Frame v.b) kullanmak o kadar kolay olur. Geniş bir kullanım alanına yönelik olarak hazırlanan bir program veya alt programda bağımsızlığa *mutlaka* büyük önem verilmesi gerekir.

Programcının bağımsız bir program hazırlaması için, kullanılan derleyici, işletim sistemi ve donanım hakkında detaylı bilgi sahibi olmak gerekir; bunlar programcının yükünü biraz daha artırır. Bir program, diğer ortamlarda mevcut olmayan alt programları veya özellikleri kullanarak, belirli bir ortamın avantajlarına dayanmamalıdır.

Bağımsız bir program hazırlamanın en güzel başlama noktası ANSI standardına uyum sağlamaktır; çünkü derleyici yazılımcılarının bir çoğu bu standartlara uyumlu derleyiciler hazırlarlar.

Genellik: Bir programın genelliği programın biraz farklı, fakat aşağı yukarı aynı işi görmesi için ne kadar kolaylıkla değiştirilebileceğinin bir ölçüsüdür. En çok üzerine düşülen nokta, girdi verilerinin nasıl temin edileceğidir. Örneğin, bir grafik çizim programı oldukça yüksek derecede genelliğe sahip olmalıdır:

- Aynı eksen takımı üzerinde birden fazla grafiğin çizilebilmesi;
- Herhangi sayıda veri girdisini her formatta kabul etmesi;
- Geniş bir aralıkta her türlü veriyi kabul etmesi;
- `INTEGER` ve `REAL` değerleri yorumlayabilmesi;
- Pozitif veya negatif değerleri de kabul etmesi;
- Eksenleri ölçeklendirebilmesi (lineer veya logaritmik ölçekte);
- Seçenek olarak eksenleri ve grafiği etiketlendirebilmesi;
- Seçenek olarak veriler arasında interpolasyon noktaları oluşturabilmesi ve interpolasyon hesabı yapması;
- Bir yazıcıya, monitöre veya bir çiziciye uyumlu çıktı üretebilmesi v.b.

Programda yukarıdaki maddelerin her birinin içermesi, belki bazı seçeneklerin asla kullanılmaması olasılığına rağmen, çok yoğun bir programlama gerektirecektir. Çalıştığı ortam için hangi seçenekleri ilave etmek veya çıkarmak programcının sorumluluğudur. Çok detaylı ve genel kapsamlı mükemmel bir program yazmak için harcanan zaman gerekli ya da gereksiz midir? Program, buna ilave edilen bu seçenekler ile, daha büyük, karmaşık ve yavaş bir program ile sonuçlanıyor mu? Bu gibi sorulara cevap aramak; cevaplar karşısında genellik sınırlarını belirlemek gerekir.

Bütünlük: Bütünlük, bir programın farklı girdi değerlerinde bile doğru sonuç verme becerisinin; bir anlamda, bir programın hatalardan ne kadar arındırıldığının ve çeşitli durumlar için ne derece test edildiğinin bir ölçüsüdür. Örneğin, sayıları büyükten küçüğe doğru dizen bir programda sayıların tümü pozitif iken başarılı bir işlem yapar; fakat negatif sayılar söz konusu olduğunda dizim işleminde hata verirse, bütünlükten yoksundur. Kullanılan bir indisli değişkenin üst sınırını aşıp aşmadığını, bölme işleminde sıfıra bölüm olup olmadığını kontrol etmeyen bir program bütünlüğe sahip değildir.

9.5 PROGRAMI BASİTLEŞTİRME

Programı basitleştirme, hem programın çalışmasını hızlandırma bakımından, hem de program yazım aşamasında daha az hata yapmak bakımından faydalı olabilmektedir. Daha önce bahsedilen hızlandırma işlemleri de basitleştirmenin bir parçası olarak sayılabilir. Buna ilaveten, bir programda yapılan işlem gurupları programın çeşitli yerlerinde tekrarlanıyorsa, tekrarlanan işlemleri gerekirse fonksiyon veya altprogramlar şeklinde hazırlamak gerekir. Bu şekilde çok sayıda tekrarlanan işlemler için *fonksiyon* ve *alt program* hazırlama şeklinde program yazma tekniğine *modüler* programlama denir. Örneğin, bir programda 8 kez farklı sayıda veri guruplarının ortalaması hesaplanıyorsa, N adet veri için ortalama hesabı yapan bir alt program bütün veri gurupları için kullanılabilir. Programda hata varsa, program daha kısa olacağından hatayı yakalamak daha kolay olur. Fakat her veri gurubunun ortalaması programda yeri geldikçe ve alt program kullanmadan yapılıyorsa, bunlardan bir veya birkaçının ortalama hesabında hata yapma olasılığı artar.

Bir matematiksel problemin çözümünde çeşitli metotları kullanırken, önce programlaması en basit olan metotlar ile yola çıkınız. Yani interpolasyon gerekli ise önce lineer interpolasyon, integral alırken yamuklar kuralını v.b programlayınız; çünkü hem formül olarak daha basit hem de programı daha kolaydır. Program tamamlandıktan sonra programınızı çalıştırarak araştırdığınız sonuç parametrelerin mantıksal olarak doğru olup olmadığını kontrol ediniz. Eğer, örneğin bir maddenin hesaplanan yoğunluğu veya bir merminin uçuş süresi negatif çıkıyorsa

programda ciddi bir hata var demektir. Fakat program doğru ve mantıksal sonuçlar veriyorsa programınızı geliştirmeye başlayabilirsiniz.

Kaç adet modül (altprogram veya fonksiyon) kullandıysanız, bunları teker teker geliştirin ve her değişiklikten sonra programınızı çalıştırıp önceki sonuç parametreler ile kıyaslayınız. Bütün modülleri aynı anda değiştirmeye kalkarsanız ve program parametreleri mantıksız bir şekilde değişirse, o zaman hangi modülde hata yaptığınızı bulmak zorlaşır.

Bazı programlama kitaplarında veya internetteki sitelerde, belirli işleri yapmak için alt programlar bulabilir; ve bu programları kendi programınızda kullanmak isteyebilirsiniz (örneğin, bu kitapta da integral, türev, v.s işlemlerini yapan altprogramlar verilmiştir). Bu durumlarda kullanmak istediğiniz programları bir kaç örnek ile test ediniz. Programın *doğru* çalıştığından ve *doğru* sonuçlar ürettiğinden emin olduktan sonra kendi programınızda kullanabilirsiniz; çünkü kitaplardaki programlar yazılırken hatalı verilebilme olasılığı vardır ve sizde programdaki hatadan habersiz olabilirsiniz.

9.6 BELLEK GEREKSİNİMİ

Bir program yazarken kullanılan değişken sayısı ve tipi, programın bellek gereksinimini belirler. Bilgisayarların anında ulaşabileceği RAM (Random Access Memory-Rastgele Erişimli Bellek) sınırlıdır. Şu andaki teknolojik sınır Windows işletim sisteminde 640 Kb'dır; ancak genişletilmiş bellek (*extended memory*) ile artırılabilmektedir. Sistem programlarının bir kısmı bilgisayar açılır açılmaz, bilgisayarda kurulu olan tüm programların bir kısmı bu belleğe yüklenir, kullanılabilir bellek alanı azalır. Bir program yazarken kullanılan değişken sayısı ve tipi, artı bazı derleyici mesajları ve komutları kullanılabilir ana bellek miktarını aşmamalıdır. Aksi takdirde program bilgisayarınızda çalışmaz.

Tablo 9.1 Değişkenlerin tiplerine göre bellek gereksinimi.

Değişken Tipi	*Bellek Gereksinimi (Byte)*
`INTEGER(KIND=1)`	1
`INTEGER(KIND=2)`	2
`INTEGER`	4
`REAL`	4
`REAL(KIND=8)`	8
`DOUBLE PRECISION`	8
`COMPLEX`	8
`COMPLEX(KIND=16)`	16
`LOGICAL`	4
`CHARACTER(LEN=m)`	m+3

Değişkenlerin tipine göre bellek alanında işgal ettiği yerler bayt (Byte) olarak Tablo 9.1'de verilmektedir. Her bir değişkenin tipiyle değişen bu miktarlardan dolayı mümkün olduğu sürece bellek gereksinimi az olan değişken tipinin kullanımı daha ekonomik olur.

Bir alfa sayısalın depolanması için gerekli bellek, alfa sayısal değişkenin uzunluğuna üç ilavesiyle bayt olarak elde edilir; yalnız alfa sayısal sabitin veya değişkenin başlama ve bitiş yerlerini belirleyen (`'` ve `“`) işaretleri bu uzunluktan sayılmaz.

Bellek gereksinimi indisli değişkenlerde de indisli değişkenin tipine göre hesaplanabilir. Örneğin, `REAL, DIMENSION(0:12)::A` deyimi ile tanımlanan `A` indisli değişkeni, tanımdan da anlaşılacağı üzere tek hassasiyetlidir; ve bellek gereksinimi 13×4=52 Bayt'tır. Fakat `DIMENSION(12)` olarak tanımlanınca indisi l'den 12'ye kadar olan 12 adet bellek alanlı bir indisli değişken oluşturulur; bu durumda da bellek gereksinimi 12×4=48 Bayt olmaktadır.

`CHARACTER(LEN=15), DIMENSION(120)::ad` şeklinde tanımlanan bir indisli alfa sayısal değişkenin bellek gereksinimi (15+3) × 121=2178 Bayt olur.

Şimdi şöyle bir durumu inceleyelim. Nüfusu 200,000 olan bir kasabadaki insanların isim ve soyadı bilgisayara girilecekse, bunları programa bir kütük ile uzunluğu 20 karakter olabilen bir indisli alfa sayısal olarak temin ettiğimizi varsayarsak, gerekli yardımcı bellek (disk alanı) gereksinimi 200,000×(20+3)=4,600,000 Bayt'tır. Bu miktar yaklaşık olarak 4.387 Mb'a (Mega bayt) karşılık gelir. Bu bilgileri indisli değişken ile bilgisayar belleğinde muhafaza etmek için, görüldüğü üzere en az 4.4 Mb ana bellek gerekecektir. Bilgisayarınızda buna yetecek genişletilmiş belleğiniz yoksa "*Bilgisayarda bu programı çalıştırmak mümkün değildir*" anlamı çıkmaktadır.

Bu problemde, ayrıca her bir şahsın cinsiyetinin de indisli alfa sayısal olarak temin edilmesi arzulanırsa, `'ERKEK'` veya `'KADIN'`ın her ikisi 5 karakter olduğundan, 200,000×8= 1,600,000 Bayt (1.526 Mb) elde edilir; oysa cinsiyet 0 ve l şeklinde (`0`=Erkek, `l`=Kadın'ı temsil edecek şekilde) bir tamsayı indisli değişkeni ile kodlanırsa 200,000×1=200,000 Bayt (0.191 Mb) elde edilerek, bellek alanı olarak % 87.5 tasarruf sağlanır. Ancak bu tamsayı tipi `INTEGER(KIND=1)` olarak tanımlanmalıdır; çünkü `INTEGER(KIND=1)` üst sınırı 127'dir.

Programlama tekniği ve bellek gereksinimi açısından, alfa sayısal değişkenler yerine, sayısal-hatta mümkünse-tamsayı değişkenler kullanılmalıdır. Örneğin, yukarıdaki problemde olduğu gibi cinsiyeti bellekte `'KADIN'` veya `'ERKEK'` gibi alfa sayısal olarak saklamak yerine tamsayı değişkenini 0 veya 1 ile temsil edebiliriz. Buna benzer şekilde, şahısların 'medeni hali' `Medeni_Hal` değişkeni, `'EVLİ'`, `'DUL'`, `'BEKAR'`, `'BOŞANMIŞ'` değerleri yerine tamsayı olarak tanımlayıp, `0` evli, `l` dul, `2` bekar ve `3` boşanmış'ı temsil edecek şekilde kodlanabilir. Yabancı dil (`'FRA'`, `'ING'`, `'ALM'` v.s) gibi veriler tamsayılar, özellikle `INTEGER(KIND=1)` (verilen tamsayı kodları 127'yi aşarsa `INTEGER(KIND=2)`'nin sayı aralığına denk gelir) olarak, ile belirtilebilir.

9.7 VERİLER VE KÜTÜK KULLANIMI

Veriler, çoğu zaman programların en can alıcı ve kaçınılmaz unsurlarıdır. Aynı zamanda program hatalarının büyük bir kısmını da verilerin programa girilmesinde ortaya çıkmaktadır. Örneğin, 20×20 boyutundaki bir matrisin 400 elemanını bir `READ*,` deyimi ile programa ekrandan girmeye kalkarsanız hata yapmanız kaçınılmazdır; ayrıca veriyi programa girme süreniz muazzam bir şekilde artar. Fakat programa veri okutma işlemini bir kütük kullanımı ile sağlarsanız, yanlış girilen bazı verileri yakalayıp düzeltmek daha kolay olur (yanlış olmayanları yeniden girmek zorunda kalmazsınız).

Binlerce sayıda veriyi sıralı erişimli kütüklerden okumak, rasgele erişimli kütükte saklanan veriyi okumaktan daha fazla zaman gerektirir. Bu nedenle gerektiğinde rasgele erişimli kütük oluşturmak çok sayıdaki veriyi okumaktan daha hızlıdır.

Ayrıca zaman zaman binlerce bazen milyonlarca veri içinden sadece bir kaçına ulaşmak istenebilir. Örneğin bir kasabada yaşayan bütün insanların isim, soyadı, adresi, doğum tarihi ve yaşının bir kütükte tutulduğunu kabul edersek, bir şahsa ait bilgilere ulaşmak için bütün bir kütüğü taramak durumunda kalabiliriz. İsmi kütüğün başına yakın bir yerde ise verilere daha kısa sürede ulaşmak mümkün olur; fakat sonunda ise daha uzun bir süre gerekir. Bu nedenle arama (bulma) işlemleri sıralı kütük kullanımı yerine rasgele-erişimli kütükler açılarak başarılmalıdır. Burada arama işlemini yapmak için uygulanan algoritmanın da arama zamanını kısa tutmak bakımından çok büyük önemi vardır.

9.8 ÇÖZÜMLEME (DEBUGGING)

Eğer her program yazıldığı ilk durumuyla çalışsaydı, programlama şu andakinden çok daha kolay bir iş olurdu. Maalesef, profesyonel tecrübeli bir programcı bile hata yapar ve program nadiren ilk hazırlandığı şekliyle çalışır. Hataları bulma ve düzeltme işlemine *çözümleme* (*debugging*) veya *hata ayıklama* adı verilir. Bazen bir programdaki hataları tespit etmek için harcanan zaman, programı yazmak için harcanan zamandan daha fazla olabilmektedir.

Programcıların yaptığı hatalar *mantık* (*programlama* ve *çalıştırma hataları* olarak ayrıca iki kısma toplanabilir) ve *sentaks hataları* olarak iki gurupta toplanır. *Sentaks hataları*, program deyimleri ve ifadelerinin FORTRAN kurallarına uymadığı durumlarda ortaya çıkar. Bunlar genellikle nokta, virgül, iki nokta üst üste, ünlem ve/veya satır devamı için & işaretinin kullanılmaması, parantez eksikliği veya fazlalığı v.s; yani daktilo hataları olarak nitelendirilen hatalar şeklinde ortaya çıkarlar. *Mantık hataları* ise programdan beklenen sonuçların elde edilmemesine neden olur. Bu hatalar ya program algoritmasının yanlış hazırlanmasından ya da algoritmanın yanlış kodlanmasından kaynaklanır. Derleyiciler program deyimlerini dikkatlice kontrol ederek yapılan sentaks hatalarını programcıya rapor eder. Mantık hataları ise, derleyicinin programcının amacını, yani ne yapmak istediğini bilmemesinden kaynaklanır ve tespiti en zor olan hatalardandır. Bu nedenle, FORTRAN kurallarını ihlal eden bir program başarı ile derlenmez; fakat, buna karşın, program deyimlerinde sentaks hatası bulunmadığı sürece mantık hatalarını tespit edemez.

9.8.1 SENTAKS HATALARI

Bu hata mesajlarından büyük bir kısmı programlama deyimlerinin yanlış olarak kullanılmasıdır. Bunlar ya programlama deyimlerinin ve kurallarının tam olarak bilinmemesinden ya da klavyeden girişte dikkatsizlik sonucunda meydana gelirler. Bunlardan `"Nested Dos with same control variable"`, `"DO increment must be non-zero"`, `"Logical constant misspelled"` v.b mesajlar programlama deyimlerinin yanlış veya eksik kullanımına örneklerdir.

Bir başka hata türü de gramer ya da *sentaks* (syntax) hatası adı verilen eksik veya fazladan parantez, fazladan nokta veya virgül kullanımı vb hatalardır. Bazı derleyiciler bu hataları düzeltebilmekte iken yapılan düzeltme programcının arzu ettiği şekilde kullanımına aykırı olabilmektedir. Sentaks hatalarına çok sık rastlanabilmesine rağmen bulunup düzeltilmesi en kolay olanlardandır.

Sentaks hataları genellikle programın çalışmasını engeller; yani programdaki sentaks hatası düzeltilmeden program çalışmaz.

Atama deyimlerinde en çok karşılaşılan hata parantezlerin eşleşmesidir. Açılan her parantez mutlaka kapatılmalıdır; yani her sol paranteze bir sağ parantez karşılık gelmelidir. Kullanılan parantez sayısı az olduğunda bu hatayı yakalamak daha kolaydır:

```
Yanlış               Doğru
X=3.14*(A-B          X=3.14*(A-B)
T=A*1.+R)**N         T=A*(1.+R)**N
```

ancak çok sayıda parantezlerde hatayı yakalamak biraz daha zorlaşır: Hatalı Deyimler

```
X=A*(B+(C-D)+E
Y=(((X**(A-3.*(C/4.))+XK-B)
Z=SQRT(1.-A/(B-1.)*(C*(SIN(B)))))
```

Burada hatayı parantezleri sayarak bulabilirsiniz. Sağ ve sol parantezlerin adedi aynı olmalıdır. Derleyici bu tür hataları yakalayacaktır; ancak formülün yazımı sırasında parantez sayısı aynı olsa bile yanlış yere yerleştirilen bir parantez ile yapılan mantık hatasını yakalamak mümkün olmamaktadır. Örneğin, $x=(a-b)/ab$ formülünü `X=(B-A)/A*B` şeklinde yazılımı yanlıştır; çünkü $x=(a-b)b/a$ ifadesine karşılık gelir. Doğru olarak yazılmış aritmetik ifade `X=(B-A)/(A*B)` şeklinde olmalıdır. Bu nedenle sentaks hatalarında formül veya sabitlerin doğru olarak girilip girilmediği dikkatlice kontrol edilmelidir.

```
X=12..35
```

Yukarıda verilen ifadede nokta (`.`) fazladan konmuştur. Program çalıştırıldığında ve bu satıra geldiğinde derleyiciler `'Syntax error in 123'` (Burada `123` hataya rastlanan satır numarası olacaktır) veya `'FATAL - Incomplete statement . '` veya `'Character expected but not found.'` gibi bir hata mesajı verir. Mesajlardaki bu farklılıklar derleyicilerin farklılıklarından kaynaklanmaktadır.

9.8.2 ÇALIŞTIRMA HATALARI (EXECUTION ERRORS)

Sentaks hatalarından farklı olarak program çalışırken, program çalışmasının bir hata mesajı ile durdurulması durumu söz konusu olabilir. Bu şekilde programın çalışmasını sona erdirmesine *çalıştırma hatası* (*execution errors* veya *run-time errors*) denir. Bu hata mesajları arasında `"Overflow ..."`, `"Underflow ..."` v.b hata mesajları sayılabilir. Bunlar programdaki sabitler eksik veya tanımsız ise (özellikle bölme işleminde tanımlanmamış değişkenin bellek değeri sıfır olduğunu hatırlayalım), programdaki değişkenlerin tiplerine göre alt ve üst sınırlarının dışına taştığında karşımıza çıkarlar. Program kimi zaman bir çıktı verirken, kimi zamanda hiç çıktı vermez yada hatalı çıktıya neden olurlar. Bu nedenle verilen hata mesajının ne olduğunun tam ve doğru olarak anlaşılması ve düzeltilmesi gereklidir.



Sentaks/Çalıştırma hata mesajları, derleyiciden derleyiciye, genellikle İngilizce olarak, verilir. Bu mesajların 'tam' olarak ne anlama geldiğini bilmek için Derleyici Kullanım Kulavuzuna başvurunuz!

9.8.3 TAMSAYI TAŞMASI (INTEGER OVERFLOW)

Tamsayı değerlerinin kullanım aralığının sınırlı olmasından daha önce bahsedilmişti. Örneğin, `INTEGER` veya `INTEGER(KIND=4)` ile tanımlanan bir değişken ($\pm2^{32}$-1) arasında değerler alabilir. Eğer programda kullanılan tamsayı, siz farkında olmadan üst sınırının dışına çıkarsa, tam sayının taşması (overflow) olarak nitelendirilen bir durum ortaya çıkar ve sonucu yanlış kılar. Programcı, hatanın farkında bile olmayabilir. En büyük tehlike büyük sayıların çok sayıda çarpılmasında ve tekrarlı toplama/çıkarma işlemlerinde karşımıza çıkar.

ÖRNEK 1: Aşağıdaki işlemi programlayarak sonucunu irdeleyiniz?

```
INTEGER :: M, K
K=3*M
PRINT*, 'M = ',M,'   K=',K
```

`K=3333333333` sayısı tamsayıların üst sınırı olan `2147483647`'den daha büyük olması `integer overflow` hata mesajına neden olur.

9.8.4 REAL OVERFLOW/ UNDERFLOW, VE SIFIRA BÖLME

Bir matematiksel işlem sonucu müsaade edilen sınırların dışına taşarsa, bu sonuç değerini bellekte depolamak mümkün olmaz; genelde değişken değeri, tipine göre, en büyük sayısal değere atanır. Eğer sayı çok büyük ise `overflow`, çok küçük ise `underflow` olarak nitelendirilen bir durum arz eder.

```
REAL :: A=1.0E+25, B=2.0E+30
C=A*B
```

olarak verilen programda `C`'nin değeri `2.0E+55` olup, müsaade edilen sınırın dışına taşmaktadır ve bu durumda işletim sistemi aracılığıyla ekrana bir hata mesajı yazılır.

Bazı durumlarda, işlem sırasının yeri değiştirilerek `overflow` veya `underflow` durumundan kaçınılabilir. Buna örnek olarak da aşağıdaki programı ele alalım:

```
REAL :: A=1.0E-30, B=2.0E-20, C=3.0E+30
D=A*B
E=D*C
```

Burada `A*B` işleminde her iki sayı müsaade edilen sınırlar içinde olmasına rağmen çarpım sınır dışına taştığından `underflow` hatası ile karşılaşılır. Oysa işlemlerin yeri aşağıdaki gibi değiştirilerek program çalıştırılabilir bir hale getirilebilir.

```
D=A*C
E=D*B
```

Çok küçük sayılara bölme işlemi yuvarlama hatalarına veya `overflow` hatasına sebep olabilir. Bu nedenle bölme işlemi yapılmadan önce bölenin sıfır veya `underflow` değerini alıp almadığının test edilmesinde fayda vardır.

Bu tür testlerden yoksun bir program mantığı mantıksal bir hataya neden olur.

9.8.5 PROGRAMLAMA HATALARI

Yukarıda bahsedilen hatalardan farklı olarak program çalışır ve sonuç üretebilir; fakat sonuçlar programlanan problemin çözümünden uzak olabilir veya en tehlikelisi de çözümün aranan çözüme yakın olmasıdır. Bu durumda programcının veya program sonuçlarını irdeleyen kişinin beklenen sonuç hakkında bir fikri olması gerekir veya modellenen fiziksel olaya uygunluğunu araştırmalıdır. Örneğin bir yayın uygulanan bir kuvvet ile hareketini veren diferansiyel denklemi verirken yayın sabitleştirildiği konumu 0 (sıfır) kabul edersek, yayın hareketi sıfır konumundan uzaklaşacak, biraz sonra sıfıra yaklaşacak ve bu hareket yavaş yavaş sönümlenen sinüzoidal bir davranış gösterecektir. Çözüm sinüzoidal harekete uymuyorsa (yani fiziksel olarak beklenen duruma aykırı ise) programda en az bir hata var demektir.

Böyle durumlarda programcı her türlü bilgisini kullanarak bu hatayı bulmak zorundadır. Yukarıda bahsettiğimiz gibi, programcının veya algoritma hazırlayanın fiziksel problemi anlaması temel kriterdir.

9.8.5.1 PROGRAMLAMA HATALARININ TESBİTİ

Bütün bunlara rağmen hatayı bulmak için aşağıdaki denemeler yapılmalıdır:

1. Program yazmaya başlamadan önce program tasarlamaya daha fazla zaman ayırın. Çoğu programcı bilgisayarın başına geçer geçmez istenen programı yazabileceğini sanır. Bu durum basit işlemlerden oluşan programların yazılmasında veya tecrübeli programcılar için geçerli olabilir. Oysa program yazmaya başladıktan kısa bir süre sonra herkes sıkıntı çekmeye başlar.
2. Yazım aşamasında programınızı bir dizi blok veya modüllere ayırın. Örneğin,
    - *Değişken Tanım Bloğu:* Programdaki bütün değişkenler ve anlamları, ve mümkünse, birimleriyle beraber listesi.
    - *Başlangıç Değeri Atama Bloğu:* Bu blokta sabitlere değer atama ve girdi verileri okunur.
    - *Hesaplama Bloğu:* Algoritmanın esasını içeren hesaplama kısmıdır.
    - *Başarı Yolu:* Programın başarılı bir şekilde sona erdiğinde buraya ulaşır ve çıktı değerlerini yazar.
    - *Başarısızlık Yolu:* Program çalışma esnasında bir problem ile karşılaşıldığında buraya ulaşır ve problemi tanımlayan mesajlar ekrana yazılır.
3. Programdaki her değişkeni, tipini (`INTEGER`, `REAL`, `CHARACTER`, v.s olarak) ve tipinin doğru tanımlayıp tanımlanmadığınızı kontrol ediniz. Bunu yapmanın en güzel yolu `IMPLICIT NONE` deyimini her programın başına yerleştirmektir. Böylece FORTRAN'ın otomatik tip belirleme kuralını kapatmış olursunuz, ve derleyici tip tanımlamasını unuttuğunuz her değişken için sizi uyarır.
4. Programa girilen verilerin değerlerini girdi işleminden hemen sonra ekrana yazdırarak, sayısal değerin doğru girilip girilmediğini kontrol ediniz.
5. Programı yazmaya başlamadan önce bir algoritma oluşturunuz veya akım şemasını çiziniz. Program başarı ile çalıştırıldıktan sonra detaylı bir akım şeması oluşturunuz.
6. Modüler programlamaya özen gösteriniz; yani programı `FUNCTION` ve `SUBROUTINE` lere ayırınız.
7. Alt programları tek tek test ediniz. Yani ikinci dereceden bir denklemin köklerini veren veya bir matrisin tersini bulan bir programı, kendiniz yazmış olabilir veya bir kitaptan veya bir arkadaşınızdan temin etmiş olabilirsiniz, bu durumda alt programı ana programdan ayırarak, bunları test ederek, gerekli işlevleri yerine getirip getirmediklerini kontrol ediniz.

Hata alt programda ise, gerekli düzeltmeleri yapınız veya hatasız ve aynı işlevi gören test edilmiş bir başka alt program kullanınız. Alt program'ları sadece bir örnek ile değil, ekstrem durumlar arzeden birkaç örnek ile test ediniz. Mesela, ikinci dereceden denklemin köklerini veren bir programı, kökler gerçek, eşit ve gerçek ve kompleks olması durumuna karşılık gelecek katsayılar girerek kontrol etmek gerekir.

8. Programlanan problemi en basit durum için çalıştırınız ve sonuçları irdeleyiniz. Bu işlem programınızın hatasız veya hatalı olduğunu göstermekten ziyade yapılmış bariz bir hatayı yakalamanıza yardımcı olabilir.
9. Program girdi değerlerini kontrol edebilirsiniz. Bazen bir girdi değerinin yanlışlıkla tanımlanmaması bütün programlama dillerinde değişken gerçek sayı ise 0, alfa sayısal ise boş karakter anlamında olacağından program akışını etkileyebilir. Özetle girdi değerlerinin hepsinin tanımlanmış olup olmadığı kontrol edilmelidir.
10. Programda fazladan çıktı deyimleri kullanarak bazı kritik program değişkenlerini ekrana veya kütüğe yazdırarak programın çalışması esnasındaki gelişimleri izlenerek, anormal değişimler varsa saptanabilir. Böylece işlem parametrelerin akışı takip edilebilir. Eğer arzu edilen çıktı sağlanamaz ise programın daha o satıra gelmeden saptığını ima eder. Bir döngü var ise acaba bu döngüyü neden yapmadı? Eğer bir blok işlemi N kez yapması gerekiyorsa, neden, mesela 6 kez, yaptı? Sayaç doğru olarak çalışıyor mu? gibi soruların ortaya çıkması ve yanıtlanması gerekebilir. Çıktı deyimlerinin ve parametrelerinin çok fazla sayıda kullanılması da şu bakımdan sakıncalı olabilir: fazla rakam arasında boğularak takibi zorlaşabilir. Özetle takip edilen değişkenlerin işlem akışı normal ise, bu değişkenlerden vazgeçip bu kez başka değişkenleri takibe almak daha doğru olur.

9.8.5.2 TİPİK SENTAKS VE/VEYA PROGRAMLAMA HATALARI

Tipik sentaks ve programlama hataları aşağıda gruplandırılmıştır:

- Karışık moda aritmetik ifadelerinin kullanımı,
- Hiçbir değerin atanmadığı değişken isminin kullanımı,
- Aritmetik ve mantıksal işlem hiyerarşisine dikkat edilmemesi,
- "Tırnak", "virgül" veya "nokta" işaretlerinin eksikliği veya fazlalığı,
- Bir deyim etiketinin veya deyim/format numarasının birden fazla kullanımı,
- Etiket veya deyim numaralarının kullanılmaması veya unutulması,
- Eşleşmeyen parantez kullanımı (açılan parantezin kapatılmaması gibi),
- Bir `INTEGER` değişken veya sabitin bir `REAL` fonksiyonun argümanı olarak kullanılması,
- Bir `REAL` değişken veya sabitin bir `INTEGER` fonksiyonun argümanı olarak kullanılması,
- Bir sayıyı `WRITE/FORMAT` deyim çifti kullanarak yazdırmak için kullanılan belirtici genişliğinin sayıyı kapsamaya yeterli olması,
- Sayaç, indeks ve DO döngüsünde artırım miktarı olarak REAL kullanımı,
- Döngü içinde sayaç, indeks ve artırım miktarının değiştirilmeye kalkışılması,
- DO döngülerinin yanlış yuvalanması,
- İlk çalıştırılabilir deyimden önce tanımlamaların yapılmaması,
- DIMENSION deyimi ile tanımlanan indisin üst sınırının aşılması.

9.8.5.3 FORTRAN DEYİMLERİNİN SIRASI

Fortran deyimleri program içinde rasgele bir düzende kullanılamaz. Temele olarak herhangi bir program veya alt programda kullanılan deyimler aşağıdaki sıralamayı takip etmelidir:

1. Program başlığı ilk satırdır (`PROGRAM`, `FUNCTION` veya `SUBROUTINE`). `PROGRAM` deyimi kullanılmayabilir ancak kullanılması tavsiye edilir.
2. Tüm belirteç deyimlere yer verilmelidir. Ayrıca FORMAT deyimlerinin hepsini bir arada vermek programa açıklık kazandırmak bakımından yararlıdır.
3. Programın algoritması gereği icra edilebilir deyimler algoritma sırasını takip etmelidir.
4. Program veya alt program END ile sonlandırılmalıdır.

Bir program veya alt programda kullanılabilecek deyimlerin genel yapısı aşağıda verilmiştir:

<table>
<tr><td colspan="3">PROGRAM, FUNCTION, SUBROUTINE, MODULE veya BLOCK DATA Deyimleri</td></tr>
<tr><td colspan="3">USE Deyimi</td></tr>
<tr><td rowspan="4">FORMAT ve ENTRY Deyimleri</td><td colspan="2">IMPLICIT NONE</td></tr>
<tr><td>PARAMETER Deyimi</td><td>IMPLICIT Deyimleri</td></tr>
<tr><td colspan="2">Türetilmiş Tip Tanımlama, Interface blokları, Tip Belirtme deyimleri,</td></tr>
<tr><td colspan="2">İcra edilebilir deyimler</td></tr>
<tr><td colspan="3">CONTAINS Deyimi</td></tr>
<tr><td colspan="3">Dahili veya Modul prosedürler</td></tr>
<tr><td colspan="3">END (PROGRAM, FUNCTION, SUBROUTINE, MODULE veya BLOCK DATA)</td></tr>
</table>

Deyim sıralamasındaki dikkatsizlikler hata mesajları ile sonuçlanır.

BÖLÜM 10
FORTRAN 90/95'İN İLERİ PROGRAMLAMA ÖZELLİKLERİ

10.1 TÜRETİLMİŞ VERİ TİPİ TANIMLAMA

Şimdiye kadar Fortran dilinin esas veri tiplerine (Real, Integer, Complex, Logical ve Character) değindik. Fortran bu veri tiplerine ilave olarak, kendi veri tiplerimizi oluşturmamıza olanak sağlar. Kullanıcı-tanımlı veri tipleri, esas veri tiplerinden yararlanarak istenilen sayıda ve kombinasyonda kullanılabilir. Kullanıcı-tanımlı veri tiplerine *türetilmiş veri tipi* denir.

Bir türetilmiş veri tipi, temel olarak, belirli bir unsur hakkındaki bilgilerin uygun bir şekilde bir araya getirilmesinden oluşur. Türetilmiş veri tipinin, aynı bir indisli değişken gibi, birden fazla bileşeni olabilir; ancak indisli değişken tek veri tipinde olmasına rağmen, türetilmiş veri bileşenlerinin tipleri farklılık gösterebilir.

Bir türetilmiş veri tipi tanımlama işlemi,TYPE-END TYPE deyim gurubu içinde bileşenlerinin ve tiplerinin tanımlanması ile gerçekleştirilir. Genel kullanım şekli

```
TYPE :: tip_ismi
      Bileşenlerin tanımlanması
      . . .
END TYPE tip_ismi
```

olarak verilmektedir, ve kullanılan bileşen sayısında herhangi bir sınırlama yoktur.

Türetilmiş veri tipleri kullanımına örnek olarak, öğrencilerin notlarının girildiği, hesaplandığı ve çıktılarının alınacağı bir program yazmayı düşünelim. Bu program için kişi hakkında bilmemiz gereken bazı bilgiler adı, soyadı, numarası, doğum tarihi, cinsiyeti vb olacaktır. Özel bir veri tipi, `kisi`, oluşturarak tüm bu verileri bir tip altında toplayabiliriz. Türetilmiş veri tipinin her unsuru *yapı* olarak isimlendirilir.

```
TYPE kisi
     CHARACTER(LEN=14) :: ad            ! 1.ci bileşen
     CHARACTER(LEN=10) :: soyad         ! 2.ci bileşen
     CHARACTER(LEN=12) :: numara        ! 3.cü bileşen
     INTEGER(KIND=1)   :: dogum_tarihi  ! 4.cü bileşen
     INTEGER           :: yas           ! 5.ci bileşen
     CHARACTER(LEN=1)  :: cinsiyet      ! 6.cı bileşen
END TYPE kisi
```

Türetilmiş veri tipi `kisi` oluşturulduktan sonra, bu tipteki değişkenler, esas tipteki değişkenler gibi, aşağıdaki şekilde tanımlanır:

```
TYPE(kisi) :: Ahmet, Fatma, Deniz
TYPE(kisi), DIMENSION(200) :: Sınıf_9FenA, Sınıf_10EdC
```

Türetilmiş verilerden oluşan sabitleri de tanımlamak mümkündür. Yapı oluşturma, tip ismini takiben, tanımlama bloğundaki sırayla bileşenlere değer atanmak suretiyle yapılır.

```
Ahmet=kisi('Ahmet','Turk','1518200x1001',1989,20,'E')
Ayse=kisi('Ayse','Gul','1518200x1002',1991,18,'K')
```

Türetilmiş veri tipi bir başka türetimiş veri tipi içinde bileşen olarak da kullanılabilir. Örneğin, not programında `not_bilgisi` isimli türetilmiş veri tipi içinde yukarıda tanımlanan `kisi` veri tipini kullanabiliriz.

```
TYPE :: not_bilgisi
     TYPE(kisi)          :: ogrenci
     INTEGER             :: Quiz_Sayisi
     REAL, DIMENSION(6):: Quiz_Notlari
     INTEGER             :: Ara_Sinav_Sayisi
     REAL, DIMENSION(3):: Ara_Sinav_Notlari
     INTEGER             :: Odev_Sayisi
     REAL, DIMENSION(9):: Odev_Notlari
     REAL                :: Final_Sinav_Notu
     REAL                :: NOT_Ortalama
END TYPE not_bilgisi
```

Tip tanımlamada, örneğin, aşağıdaki şekilde kullanabiliriz:

```
TYPE(not_bilgisi) , DIMENSION(90) :: SINIF_1, SINIF_2
```

10.2 TÜRETİLMİŞ VERİ TİPLERİNİN KULLANIMI

Türetilmiş veri tipi değişkenlerinin bileşenlerine diğer bileşenlerden bağımsız olarak erişilebilir ve aynı tipteki başka değişkenler gibi kullanılabilir. Tip bileşeni, bileşen seçici (% işareti) ile kullanılır. Türetilmiş veri ismini takiben % işareti ve bunu da takiben bileşen ismi yazılır. Örneğin,

```
Ahmet%adi='Ahmet'
Ahmet%numara='1518200x1001'
SINIF_1%NOT_Ortalama=67
SINIF_2(51)%Odev_sayisi=7
```

ÖRNEK 1: A(x_0, y_0, z_0) ile B(x_1, y_1, z_1) noktaları arasındaki uzaklığı veren türetilmiş veri tipi kullanımını içeren bir program yazalım. AB noktaları arasındaki uzaklık

$$d = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2}$$

bağıntısıyla verilmektedir.

Kartezyen koordinat sistemindeki bir nokta (*x*,*y*,*z*) üçlüsü ile belirlendiğine göre, bir noktanın konumunu türetilmiş veri tipi olarak tanımlayabiliriz.

```
PROGRAM Ornek1
IMPLICIT NONE
  TYPE konum           !  Konum isimli veri tipi oluşturma
     REAL :: x         !  1.ci bileşen
     REAL :: y         !  2.ci bileşen
     REAL :: z         !  3.cü bileşen
  END TYPE konum
!
TYPE (konum):: a,b   ! a ve b değişkenlerini konum tipinde tanımlama
REAL :: uzaklik
!
 a = konum(3.,-2., 4.)
 b = konum(1., 4.,-3.)
PRINT 25, a
PRINT 25, b
uzaklik=SQRT((a%x-b%x)**2+(a%y-b%y)**2+(a%z-b%z)**2)
PRINT *, ' '
PRINT 25, uzaklık
25 FORMAT(3x,3(f7.3,2x))
END PROGRAM Ornek1
```

a ve b değişkenleri konum tipinde tanımlandıktan sonra, a ve b'nin bileşenleri uzaklık bağıntısında %x, %y ve %z ekleri kullanılarak yazılırlar. Programının çıktısı

```
    3.000   -2.000    4.000
    1.000    4.000   -3.000

    9.434
```

olarak elde edilir.

ÖRNEK 2: Türetilmiş veri tipi olarak tanımlanan karmaşık sayılarla toplama ve çarpma işlemi yapan bir program yazınız.

Esasen karmaşık sayılarla işlem yapmak için bu tarz bir program yazmamıza gerek yoktur çünkü COMPLEX olarak tanımlanan karmaşık sayılarla aritmetik işlem operatörlerini kolaylıkla kullanabiliriz. Türetilmiş veri tipi tanımlama ve kullanmayı açıklamak için bir karmaşık sayıyı sanal = x + **i** y şeklinde tanımlarız.

```
PROGRAM Ornek2
IMPLICIT NONE
     TYPE sanal        ! sanal = x + i y   olarak Tip tanımı
       REAL :: x
       REAL :: y
     END TYPE sanal
TYPE (sanal):: z1,z2,z3,z4 ! sanal değişkenler olarak tanımlanıyor
```

```
z1 = sanal(3.,-2.)          ! z1 = 3 - 2 i
z2 = sanal(1., 4.)          ! z2 = 1 + 4 i sabitleri tanımlandı
! z3=z1+z2 işlemi
z3%x=z1%x+z2%x
z3%y=z1%y+z2%y
! z4=z1*z2 işlemi
z4%x=z1%x*z2%x-z1%y*z2%y
z4%y=z1%x*z2%y+z1%y*z2%x
! Bu programda z3=z1+z2 veya z4=z1*z2 şeklinde kullanamayız!
PRINT 25, z1
PRINT 25, z2
PRINT *, '    z1 + z2 '
PRINT 25, z3
PRINT *, '    z1 * z2 '
PRINT 25, z4
25 FORMAT(3x,3(f5.1,2x))
END PROGRAM Ornek2
```

Programının çıktısı

```
   3.0   -2.0
   1.0    4.0
   z1 + z2
   4.0    2.0
   z1 * z2
  11.0   10.0
```

ÖRNEK 3: Kişisel (ad, soyad, numara, doğum tarihi, yaş ve cinsiyeti) bilgilerinin türetilmiş veri tipi `kisi` olarak tanımlandığı ve kullanıldığı bir program ele alalım.

```
PROGRAM Ornek3
IMPLICIT NONE
     TYPE kisi   ! Türetilmiş tip tanım bloğu
          CHARACTER(LEN=14) :: ad
          CHARACTER(LEN=10) :: soyad
          CHARACTER(LEN=12) :: numara
          INTEGER           :: dogum_tarihi
          INTEGER           :: yas
          CHARACTER(LEN=1)  :: cinsiyet
     END TYPE kisi
TYPE (kisi) :: Ahmet   ! Türetilmiş tipteki değişken tanımı
! Değer atama
ahmet = kisi('Ahmet','Türk','1518200x1001',1989,21,'E')
WRITE(*,*) 'Serbest format: ',Ahmet ! Serbest formatta çıktı alma
WRITE(*,10) Ahmet                   ! Formatlı çıktı alma
10 FORMAT('Formatlı çıktı:',/,3(1X,A,/),1X,i5,/,1x,i4,1x,A)
END PROGRAM Ornek3
```

Programının çıktısı

```
Serbest format: Ahmet         Türk      1518200x1001 1989 21 E
Formatlı çıktı:
Ahmet
Türk
1518200x1001
 1989
  21 E
```

Girdi-çıktı deyimlerinde türetilmiş veri bileşenlerinin tiplerine uygun format deyimleri kullanılmalıdır.

10.3 JENERİK ALT PROGRAMLAR

FORTRAN 90/95 programlama dili hem arşiv hem de jenerik fonksiyonlar içerir. *Jenerik fonksiyon*, farklı tipteki veri girdileriyle çalışabilen programlardır. Örneğin, mutlak değer fonksiyonu olan `ABS()`, tamsayılar için `IABS()`, çift hassasiyetli gerçek sayılar için `DABS()`, karmaşık sayılar `CABS()` vb isimlerini alır. `ABS()` fonksiyonunu herhangi bir veri tipi için kullanabiliriz çünkü Fortran derleyicisi girdi tipine bakarak tam sayı ise `IABS`, çift hassasiyetli gerçek sayı ise `DABS` vs programlarını kullanır.

10.3.1 KULLANICI-TANIMLI JENERİK ALT PROGRAMLAR

FORTRAN 90/95 ve 2003'de kullanıcının kendi jenerik alt programlarını oluşturmasına olanak sağlanır. Örneğin sıralama işlemi yapan bir alt program oluşturmak istersek, "bu sıralamayı hangi veri tipi için yapmalıyız?" sorusunun cevabı: `Integer`, `Real`, `Real*8`, `Character` tipleri olabilir. Böylece dört veri türüne göre sıralama yapan alt programlar hazırlayabiliriz. Bu işlemi özel bir arayüz bloğu oluşturarak uygulayabiliriz. Genel kullanım şekli

```
INTERFACE jenerik_isim
     Özel_arayüz_1
     Özel_arayüz_2
     . . .
END INTERFACE
```

olarak verilmektedir. Böylece derleyici jenerik alt program ile karşılaştığında, alt programın arayüzünde tanımlı girdi tipleri ile karşılaştırma yapar ve girdisi uygun olan programı kullanır. Jenerik alt programların kullanımında aşağıdaki kurallara dikkat edilmelidir:

1. Jenerik arayüz bloğunda tanımlanan alt programların tamamı ya `FUNCTION` ya da `SUBROUTINE` olmalıdır. Alt program türleri karışımı kabul edilmez.

2. Her alt program, diğer alt programlardan blok tipi, sayısı, konumu, opsiyonel argümanların olup olmaması ile ayırt edilmelidir. Böylece derleyici hangi programı kullanacağında bir karmaşa yaşamaz.

Örneğin, bir programcı aşağıda verilen, büyükten küçüğe doğru sıralama yapan, dört program yazmış olsun.

Alt Program	**İşlevi**
SUBROUTINE SIRALA**T**(deger,indisli)	Tamsayı verileri sıralamak
SUBROUTINE SIRALA**G**(deger,indisli)	Tek hassasiyetli gerçek verileri sıralamak
SUBROUTINE SIRALA**C**(deger,indisli)	Çift hassasiyetli gerçek verileri sıralamak
SUBROUTINE SIRALA**A**(deger,indisli)	Alfa sayısal verileri sıralamak

Şimdi `SIRALA` isimli bir jenerik `SUBROUTINE` alt programı oluşturmak isteyelim. Bu işlem aşağıda verilen `INTERFACE` arayüz bloğu ile gerçekleştirilebilir.

```
INTERFACE SIRALA

     SUBROUTINE SIRALAT(deger,indisli)
       IMPLICIT NONE
       INTEGER, INTENT(IN) :: deger
       INTEGER, INTENT(INOUT) :: indisli
     END SUBROUTINE SIRALAT
     SUBROUTINE SIRALAG(deger,indisli)
       IMPLICIT NONE
       INTEGER, INTENT(IN) :: deger
       REAL, INTENT(INOUT) :: indisli
     END SUBROUTINE SIRALAG
     SUBROUTINE SIRALAC(deger,indisli)
       IMPLICIT NONE
       INTEGER, INTENT(IN) :: deger
       REAL*8, INTENT(INOUT) :: indisli
     END SUBROUTINE SIRALAC
     SUBROUTINE SIRALAA(deger,indisli)
       IMPLICIT NONE
       INTEGER, INTENT(IN) :: deger
       CHARACTER(LEN=*), INTENT(INOUT) :: indisli
     END SUBROUTINE SIRALAA

END INTERFACE SIRALA
```

Bu şekilde tanımlanan jenerik alt programlar, programda `CALL SIRALA(..)` ile karşılaşıldığında, derleyici `indisli` değişkeninin tipine göre uygun alt programı seçer ve kullanır.

10.3.2 MODÜLLERDE ALT PROGRAMLAR İÇİN JENERİK ARAYÜZ KULLANIMI

Önceki örnekte, her alt program için tanımlarla beraber açık arayüz verilmişti. Arayüzde yer alan her alt program ayrı ayrı derlenmiş olursa ve tanımlama blokları yer almaz ise, bu şekildeki kullanımın pek yararı olmaz. Ancak her alt program bir modül içinde yer aldığında ve açık arayüzleri mevcut olursa, durum farklı olur. Dolayısıyla bu problemin üstesinden gelmek için

Fortran diline `MODULE PROCEDURE` deyimi ilave edilmiştir. Genel kullanım şekli

```
MODULE PROCEDURE modül_altprogram1 (,modül_altprogram2,..)
```

olarak verilir ve birden fazla modül alt programları peş peşe sıralanabilir. Bu alt programlara `USE` deyimi ile ilgili alt veya ana programdan erişilebilir.

Sıralama işlemini yapan jenerik alt program modül alt program olarak tanımlansaydı, aşağıdaki şekilde verilmesi gerekirdi.

```
INTERFACE SIRALA
     MODULE PROCEDURE SIRALAT
     MODULE PROCEDURE SIRALAG
     MODULE PROCEDURE SIRALAC
     MODULE PROCEDURE SIRALAA
END INTERFACE SIRALA
```

ÖRNEK 4: Tamsayı, gerçek sayı, çift hassasiyetli gerçek, karmaşık veya karmaşık çift hassasiyetli sayılardan oluşabilen indisli bir değişkenin en büyük değerini ve opsiyonel olarak konumunu veren bir `En_Buyuk` isimli bir `SUBROUTINE` alt programı yazınız. (Karmaşık sayılarda mutlak değerce en büyük olan sayı araştırılacaktır!)

Aşağıda `MODULE` içinde modül alt programların tanımlamaları ve programlar verilmektedir.

```
MODULE jenerik_En_Buyuk
IMPLICIT NONE
! Her alt programda kullanılan sabitlerin tanımlanması
INTEGER, PARAMETER :: tek  = SELECTED_REAL_KIND(p=6)
INTEGER, PARAMETER :: cift = SELECTED_REAL_KIND(p=13)
! Modül prosedür olarak tanımlanması
INTERFACE En_Buyuk
   MODULE PROCEDURE En_Buyuk_i    ! Tamsayılar için
   MODULE PROCEDURE En_Buyuk_r    ! Gerçek sayılar için
   MODULE PROCEDURE En_Buyuk_d    ! Çift hassasiyetli sayılar için
   MODULE PROCEDURE En_Buyuk_c    ! Karmaşık sayılar için
   MODULE PROCEDURE En_Buyuk_dc   ! Çift hassasiyetli karmaşık sayılar için
END INTERFACE
! *********  Jenerik Alt programların verilmesi  ******************
CONTAINS
   SUBROUTINE En_Buyuk_i ( indisli, deger, BuyukDeger, Konum )
   !  TAMSAYILAR İÇİNDE EN BÜYÜK DEĞER VE KONUMUNU VEREN ALT PROGRAM
   !   Değişken Listesi ve Tanımları
   !   Deger            :  indisli değişkenin boyutu
   !   Indisli(Deger)   :  Deger boyutunda indisli değişken
   !   BuyukDeger       :  En büyük değer
   !   Konum            :  En büyük değerin konumu
   !
   IMPLICIT NONE
```

```
INTEGER, INTENT(IN) :: deger
INTEGER, INTENT(IN), DIMENSION(deger) :: indisli
INTEGER, INTENT(OUT) :: BuyukDeger
INTEGER, INTENT(OUT), OPTIONAL :: Konum
INTEGER :: i
INTEGER :: max_konum
BuyukDeger = indisli(1)
max_konum = 1
DO i = 2, deger
   IF ( indisli(i) > BuyukDeger ) THEN
      BuyukDeger = indisli(i)
      max_konum = i
   END IF
END DO
IF ( PRESENT(Konum) ) THEN
   Konum = max_konum
END IF
END SUBROUTINE En_Buyuk_i

SUBROUTINE En_Buyuk_r ( indisli, deger, BuyukDeger, Konum )
!  GERÇEK SAYILAR İÇİNDE EN BÜYÜK DEĞER VE KONUMUNU VEREN
!  ALT PROGRAM
IMPLICIT NONE
INTEGER, INTENT(IN) :: deger
REAL(KIND=tek), INTENT(IN), DIMENSION(deger) :: indisli
REAL(KIND=tek), INTENT(OUT) :: BuyukDeger
INTEGER, INTENT(OUT), OPTIONAL :: Konum
INTEGER :: i
INTEGER :: max_konum
BuyukDeger = indisli(1)
max_konum = 1
DO i = 2, deger
   IF ( indisli(i) > BuyukDeger ) THEN
      BuyukDeger = indisli(i)
      max_konum = i
   END IF
END DO
IF ( PRESENT(Konum) ) THEN
   Konum = max_konum
END IF
END SUBROUTINE En_Buyuk_r

SUBROUTINE En_Buyuk_d ( indisli, deger, BuyukDeger, Konum )
!  ÇİFT HASSASİYETLİ GERÇEK SAYILAR İÇİNDE EN BÜYÜK DEĞER
!  VE KONUMUNU VEREN ALT PROGRAM
IMPLICIT NONE
INTEGER, INTENT(IN) :: deger
REAL(KIND=cift), INTENT(IN), DIMENSION(deger) :: indisli
REAL(KIND=cift), INTENT(OUT) :: BuyukDeger
INTEGER, INTENT(OUT), OPTIONAL :: Konum
INTEGER :: i
INTEGER :: max_konum
BuyukDeger = indisli(1)
max_konum = 1
DO i = 2, deger
   IF ( indisli(i) > BuyukDeger ) THEN
```

```
         BuyukDeger = indisli(i)
         max_konum = i
      END IF
   END DO
   IF ( PRESENT(Konum) ) THEN
      Konum = max_konum
   END IF
   END SUBROUTINE En_Buyuk_d

   SUBROUTINE En_Buyuk_c ( indisli, deger, BuyukDeger, Konum )
   !  KARMAŞIK SAYILAR İÇİNDE EN BÜYÜK DEĞER VE KONUMUNU VEREN
   !  ALT PROGRAM
   IMPLICIT NONE
   INTEGER, INTENT(IN) :: deger
   COMPLEX(KIND=tek), INTENT(IN), DIMENSION(deger) :: indisli
   REAL(KIND=tek), INTENT(OUT) :: BuyukDeger
   INTEGER, INTENT(OUT), OPTIONAL :: Konum
   INTEGER :: i
   INTEGER :: max_konum
   BuyukDeger = ABS(indisli(1))
   max_konum = 1
   DO i = 2, deger
      IF ( ABS(indisli(i)) > BuyukDeger ) THEN
         BuyukDeger = ABS(indisli(i))
         max_konum = i
      END IF
   END DO
   IF ( PRESENT(Konum) ) THEN
      Konum = max_konum
   END IF
   END SUBROUTINE En_Buyuk_c

   SUBROUTINE En_Buyuk_dc ( indisli, deger, BuyukDeger, Konum )
   !  KARMAŞIK (çift hassasiyetli) SAYILAR İÇİNDE EN BÜYÜK DEĞER
   !  VE KONUMUNU VEREN ALT PROGRAM
   IMPLICIT NONE
   INTEGER, INTENT(IN) :: deger
   COMPLEX(KIND=cift), INTENT(IN), DIMENSION(deger) :: indisli
   REAL(KIND=cift), INTENT(OUT) :: BuyukDeger
   INTEGER, INTENT(OUT), OPTIONAL :: Konum
   INTEGER :: i
   INTEGER :: max_konum
   BuyukDeger = ABS(indisli(1))
   max_konum = 1
   DO i = 2, deger
      IF ( ABS(indisli(i)) > BuyukDeger ) THEN
         BuyukDeger = ABS(indisli(i))
         max_konum = i
      END IF
   END DO
   IF ( PRESENT(Konum) ) THEN
      Konum = max_konum
   END IF
   END SUBROUTINE En_Buyuk_dc

END MODULE jenerik_En_Buyuk
```

```
PROGRAM Ornek4
!                          Test için kullanılan ana program
USE jenerik_En_Buyuk  !  Ana programdan modüle erişim
IMPLICIT NONE
INTEGER, DIMENSION(6) :: indisli_i
! İndisli değişken tanımlamaları
REAL(KIND=tek ), DIMENSION(6) :: indisli_r
REAL(KIND=cift), DIMENSION(6) :: indisli_d
COMPLEX(KIND=tek ), DIMENSION(6) :: indisli_c
COMPLEX(KIND=cift), DIMENSION(6) :: indisli_dc
INTEGER :: EnBuyuk_Deger_i
REAL(KIND=tek ) :: EnBuyuk_Deger_r
REAL(KIND=cift) :: EnBuyuk_Deger_d
INTEGER :: Konum_EnBuyuk
! Değişkenlere değer atama
indisli_i =(/ -21, 13, 42,  0,  55,  -29 /)
indisli_r =(/ -23.,33.,82.,59., 95., -32./)
indisli_d =(/ -13._cift, 3._cift, 2._cift, 0._cift, 25._cift, -2._cift
/)
indisli_c=(/ (3.,2.), (-4.,-5.), (3.,-9), (1.,6.), &
              (-5.,11.), (8.,-4.) /)
indisli_dc=(/ (1._cift,3._cift), (-2._cift,-4._cift), &
              (4._cift,1._cift), (-4._cift,5._cift), &
              (6._cift,-3._cift), (2._cift,-5._cift) /)
CALL En_Buyuk(indisli_i, 6, EnBuyuk_Deger_i, Konum_EnBuyuk )
  WRITE (*,10) EnBuyuk_Deger_i, Konum_EnBuyuk
CALL En_Buyuk(indisli_r, 6, EnBuyuk_Deger_r )
  WRITE (*,20) EnBuyuk_Deger_r
CALL En_Buyuk(INDISLI=indisli_d,DEGER=6,BUYUKDEGER=EnBuyuk_Deger_d)
  WRITE (*,30) EnBuyuk_Deger_d
CALL En_Buyuk(DEGER=6,INDISLI=indisli_c,BUYUKDEGER=EnBuyuk_Deger_r, &
              KONUM=Konum_EnBuyuk )
  WRITE (*,40) EnBuyuk_Deger_r, Konum_EnBuyuk
CALL En_Buyuk(indisli_dc, 6, EnBuyuk_Deger_d)
  WRITE (*,50) EnBuyuk_Deger_d
10 FORMAT(' Tamsayı girdiler: '/,' En Büyük Değer = ',I3, &
   ';  Konumu = ', I3 )
20 FORMAT(' Tek Hassasiyetli Gerçek Sayılı Girdiler : '/, &
   ' En Büyük Değer = ',F7.3)
30 FORMAT(' Çift Hassasiyetli Gerçek Sayılı Girdiler : '/, &
   ' En Büyük Değer = ',F7.3)
40 FORMAT(' Tek Hassasiyetli karmaşık sayılı Girdiler '/, &
   ' En Büyük Değer = ',F7.3,';  konumu = ', I3 )
50 FORMAT(' Çift Hassasiyetli karmaşık sayılı Girdiler '/, &
   ' En Büyük Değer = ',F7.3)
END PROGRAM Ornek4
```

Programının çıktısı

```
Tamsayı girdiler:
En Büyük Değer =  55;  Konumu =   5
Tek Hassasiyetli Gerçek Sayılı Girdiler :
En Büyük Değer =  95.000
Çift Hassasiyetli Gerçek Sayılı Girdiler :
En Büyük Değer =  25.000
Tek Hassasiyetli karmaşık sayılı Girdiler
En Büyük Değer =  12.083;  konumu =   5
Çift Hassasiyetli karmaşık sayılı Girdiler
En Büyük Değer =  10.296
```

10.3.3 KULLANICI-TANIMLI İŞLEM OPERATÖRLERİ VE ATAMALAR

Türetilmiş veri tipleri ile yapılan işlemlerde aritmetik ve mantıksal işlem operatörleri kullanılamaz. Örneğin, türetilmiş veri tipi olarak tanımlanan iki karmaşık sayıyı toplama, çıkarma, çarpma işlemlerini **z1+z2**, **z1*z2** vb şeklinde yapamayız; işlemlerin her bir kademesini ayrıntılı olarak belirtmemiz ve programlamamız gerekir. Ancak Fortran arşiv programları COMPLEX tanımlaması ile karmaşık sayılarla aritmetik işlem yapmaya olanak sağlar.

Kullanıcı-tanımlı türetilmiş veri tiplerini aritmetik veya mantıksal işlem operatörleri ile kullanmak istersek arayüz operatörleri tanımlamamız gerekir. Bu amaçla kullanılan arayüz operatör bloğu genel şekli

```
INTERFACE OPERATOR (operatör_sembolü)
     MODULE PROCEDURE fonksiyon_1
     . . .
END INTERFACE
```

olarak verilmektedir.

Operatör sembolü standart (+,-,/,*,>,<, vs) olabileceği gibi kullanıcı-tanımlı başka operatörler de olabilir. Kullanıcı-tanımlı operatör ismi nokta işaretleri arasında 31 karakter uzunluğunda isim alabilir ve "_" işareti ile rakamlar kullanılamaz!
Fortran aritmetik işlem operatörleri kullanılıyorsa, dikkat edilmesi gereken noktalar şunlardır:

1. İşlem operatörünün anlamını değiştirilemez. Örneğin, '+' toplama operatörü çıkarma işlemi veya bölme işlemi amacıyla kullanılamaz.
2. Bir işlevi görmek için gerekli argüman sayısı operatörün normal kullanımındaki argüman sayısıyla eş olmalıdır. Örneğin, '+' operatörü işlemi için iki değişken gerekir.
3. İlişkisel bir operatör kullanıldığında, anlamı değiştirilemez. '<' küçük sembolü anlamı aynıdır.

ÖRNEK 5: Boyutu 3 olan indisli değişkeni vektöre, vektörü indisli değişkene dönüştüren, vektörler arasında aritmetik işlemler (toplama, çıkarma, sabit ile çarpma, sabite bölme), skalar ve vektörel çarpım işlemlerini yapan kullanıcı-tanımlı operatörler hazırlayınız.

Bu programı oluşturmak için aşağıdaki tabloda verilen alt programları oluşturacağız:

Alt Program	Girdi(ler) ve Tipi	Çıktı(lar) ve Tipi	Alt program tipi
`indisliden_vektore`	Gerçek 3 elemanlı indisli değişken		Vektör
`vektorden_indisliye`	Vektör		Gerçek 3 elemanlı indisli değişken
`vektor_topla`	Vektör	Vektör	Vektör
`vektor_cikarma`	Vektör	Vektör	Vektör
`vektor_carpi_reelS`	Vektör	Gerçek sayı	Vektör
`reelS_carpi_vektor`	Gerçek sayı	Vektör	Vektör
`vektor_carpi_tamsayi`	Vektör	Tamsayı	Vektör
`tamsayi_carpi_vektor`	Tamsayı	Vektör	Vektör
`vektorel_carpim`	Vektör	Vektör	Vektör
`vektor_bolu_reel`	Vektör	Gerçek sayı	Vektör
`vektor_bolu_tamsayi`	Vektör	Tamsayı	Vektör
`skalar_carpim`	Vektör	Vektör	Gerçek sayı

Kullanılan değişken ve/veya sabitler

`Vektor_1, Vektor_2` : Vektörler
`İndisli` : İndisli değişken [`indisli(3)`]
`Gercek_1, Gercek_2` : Tek hassasiyetli gerçek sayı sabitleri
`Tamsayi_1, Tamsayi_2`: Tamsayı sabitleri

Programda normalde kabul edilmeyen vektörlerle gerçek ve tamsayıların çarpılması, bölünmesi, skalar ve vektörel çarpım normal aritmetik işlemler ile kullanılabilir hale getirilmektedir.

```
MODULE vektorler
!      İşlem                                         Operatör
!      =====                                         ========
!  1.  Gerçek indisli değişkenden vektör oluşturma      =
!  2.  Vektördem indisli değişken oluşturma             =
!  3.  Vektör toplama işlemi                            +
!  4.  Vektör çıkarma işlemi                            -
!  5.  Vektör-skalar sabit çarpımı                      *
!  6.  Vektör-skalar sabit bölümü                       /
!  7.  Skalar çarpım                                 .SCALAR.
!  8.  Vektörel çarpım                                  *
IMPLICIT NONE
TYPE :: vektor
   REAL :: x
   REAL :: y
   REAL :: z
END TYPE

INTERFACE ASSIGNMENT (=)   ! Eşitlik, denklik operatörü
   MODULE PROCEDURE indisliden_vektore
   MODULE PROCEDURE vektorden_indisliye
END INTERFACE
```

```
INTERFACE OPERATOR (+)     ! İki vektör toplamı
   MODULE PROCEDURE vektor_topla
END INTERFACE

INTERFACE OPERATOR (-)     ! İki vektör farkı
   MODULE PROCEDURE vektor_cikarma
END INTERFACE

INTERFACE OPERATOR (*)     ! Vektör skalar sabit çarpımları
   MODULE PROCEDURE vektor_carpi_reelS
   MODULE PROCEDURE reelS_carpi_vektor
   MODULE PROCEDURE vektor_carpi_tamsayi
   MODULE PROCEDURE tamsayi_carpi_vektor
   MODULE PROCEDURE vektorel_carpim
END INTERFACE

INTERFACE OPERATOR (/)     ! Vektör skalar sabit bölme işlemleri
   MODULE PROCEDURE vektor_bolu_reel
   MODULE PROCEDURE vektor_bolu_tamsayi
END INTERFACE

INTERFACE OPERATOR (.SKALAR.)    ! İki vektörün skalar çarpımı
   MODULE PROCEDURE skalar_carpim
END INTERFACE

CONTAINS
   SUBROUTINE indisliden_vektore(vektor_sonuc, indisli)
      TYPE (vektor), INTENT(OUT) :: vektor_sonuc
      REAL, DIMENSION(3), INTENT(IN) :: indisli
      vektor_sonuc%x=indisli(1)
      vektor_sonuc%y=indisli(2)
      vektor_sonuc%z=indisli(3)
   END SUBROUTINE indisliden_vektore

   SUBROUTINE vektorden_indisliye(indisli_result, vektor_1)
      REAL, DIMENSION(3), INTENT(OUT) :: indisli_result
      TYPE (vektor), INTENT(IN) :: vektor_1
      indisli_result(1)=vektor_1%x
      indisli_result(2)=vektor_1%y
      indisli_result(3)=vektor_1%z
   END SUBROUTINE vektorden_indisliye

   FUNCTION vektor_topla(vektor_1, vektor_2)
      TYPE (vektor) :: vektor_topla
      TYPE (vektor), INTENT(IN) :: vektor_1, vektor_2
      vektor_topla%x=vektor_1%x+vektor_2%x
      vektor_topla%y=vektor_1%y+vektor_2%y
      vektor_topla%z=vektor_1%z+vektor_2%z
   END FUNCTION vektor_topla

   FUNCTION vektor_cikarma(vektor_1, vektor_2)
      TYPE (vektor) :: vektor_cikarma
      TYPE (vektor), INTENT(IN) :: vektor_1, vektor_2
      vektor_cikarma%x=vektor_1%x-vektor_2%x
      vektor_cikarma%y=vektor_1%y-vektor_2%y
      vektor_cikarma%z=vektor_1%z-vektor_2%z
```

```
END FUNCTION vektor_cikarma

FUNCTION vektor_carpi_reelS(vektor_1, Gercek_2)
   TYPE (vektor) :: vektor_carpi_reelS
   TYPE (vektor), INTENT(IN) :: vektor_1
   REAL, INTENT(IN) :: Gercek_2
   vektor_carpi_reelS%x=vektor_1%x*Gercek_2
   vektor_carpi_reelS%y=vektor_1%y*Gercek_2
   vektor_carpi_reelS%z=vektor_1%z*Gercek_2
END FUNCTION vektor_carpi_reelS

FUNCTION reelS_carpi_vektor(Gercek_1, vektor_2)
   TYPE (vektor) :: reelS_carpi_vektor
   REAL, INTENT(IN) :: Gercek_1
   TYPE (vektor), INTENT(IN) :: vektor_2
   reelS_carpi_vektor%x=Gercek_1*vektor_2%x
   reelS_carpi_vektor%y=Gercek_1*vektor_2%y
   reelS_carpi_vektor%z=Gercek_1*vektor_2%z
END FUNCTION reelS_carpi_vektor

FUNCTION vektor_carpi_tamsayi(vektor_1, Tamsayi_2)
   TYPE (vektor) :: vektor_carpi_tamsayi
   TYPE (vektor), INTENT(IN) :: vektor_1
   INTEGER, INTENT(IN) :: Tamsayi_2
   vektor_carpi_tamsayi%x=vektor_1%x*REAL(Tamsayi_2)
   vektor_carpi_tamsayi%y=vektor_1%y*REAL(Tamsayi_2)
   vektor_carpi_tamsayi%z=vektor_1%z*REAL(Tamsayi_2)
END FUNCTION vektor_carpi_tamsayi

FUNCTION tamsayi_carpi_vektor(Tamsayi_1, vektor_2)
   TYPE (vektor) :: tamsayi_carpi_vektor
   INTEGER, INTENT(IN) :: Tamsayi_1
   TYPE (vektor), INTENT(IN) :: vektor_2
   tamsayi_carpi_vektor%x=REAL(Tamsayi_1)*vektor_2%x
   tamsayi_carpi_vektor%y=REAL(Tamsayi_1)*vektor_2%y
   tamsayi_carpi_vektor%z=REAL(Tamsayi_1)*vektor_2%z
END FUNCTION tamsayi_carpi_vektor

FUNCTION vektor_bolu_reel(vektor_1, Gercek_2)
   TYPE (vektor) :: vektor_bolu_reel
   TYPE (vektor), INTENT(IN) :: vektor_1
   REAL, INTENT(IN) :: Gercek_2
   vektor_bolu_reel%x=vektor_1%x/Gercek_2
   vektor_bolu_reel%y=vektor_1%y/Gercek_2
   vektor_bolu_reel%z=vektor_1%z/Gercek_2
END FUNCTION vektor_bolu_reel

FUNCTION vektor_bolu_tamsayi(vektor_1, Tamsayi_2)
   TYPE (vektor) :: vektor_bolu_tamsayi
   TYPE (vektor), INTENT(IN) :: vektor_1
   INTEGER, INTENT(IN) :: Tamsayi_2
   vektor_bolu_tamsayi%x=vektor_1%x/REAL(Tamsayi_2)
   vektor_bolu_tamsayi%y=vektor_1%y/REAL(Tamsayi_2)
   vektor_bolu_tamsayi%z=vektor_1%z/REAL(Tamsayi_2)
END FUNCTION vektor_bolu_tamsayi
```

```
   FUNCTION skalar_carpim(vektor_1, vektor_2)
      REAL :: skalar_carpim
      TYPE (vektor), INTENT(IN) :: vektor_1, vektor_2
      skalar_carpim = vektor_1%x*vektor_2%x + &
         vektor_1%y*vektor_2%y+vektor_1%z*vektor_2%z
   END FUNCTION skalar_carpim

   FUNCTION vektorel_carpim(vektor_1, vektor_2)
      TYPE (vektor) :: vektorel_carpim
      TYPE (vektor), INTENT(IN) :: vektor_1, vektor_2
      vektorel_carpim%x=vektor_1%y*vektor_2%z-vektor_1%z*vektor_2%y
      vektorel_carpim%y=vektor_1%z*vektor_2%x-vektor_1%x*vektor_2%z
      vektorel_carpim%z=vektor_1%x*vektor_2%y-vektor_1%y*vektor_2%x
   END FUNCTION vektorel_carpim
END MODULE vektorler

PROGRAM Ornek5
USE vektorler  ! Vektorler atama ve operator programlarının çağırılması
IMPLICIT NONE
REAL, DIMENSION(3) :: indisli_cikti
TYPE (vektor) :: vektor_1, vektor_2
vektor_1 = (/ 1., 2., 3. /)
indisli_cikti = vektor_1
WRITE (*,10) vektor_1, indisli_cikti
vektor_1 = (/ 10., 20., 30. /)
vektor_2 = (/ 1., 2., 3. /)
WRITE (*,20) vektor_1,vektor_2,vektor_1+vektor_2,vektor_1-vektor_2
vektor_1 = (/ 1., 2., 3. /)
WRITE (*,30) vektor_1,2.*vektor_1,vektor_1*2.,2*vektor_1,vektor_1*2
vektor_1 = (/ 10., 20., 30. /)
WRITE (*,40) vektor_1,vektor_1/5.,vektor_1/5
vektor_1 = (/ 1., 2., 3. /)
vektor_2 = (/ 1., 2., 3. /)
WRITE (*,50) vektor_1,vektor_2,vektor_1.SKALAR.vektor_2
vektor_1 = (/ 1., -1., 1. /)
vektor_2 = (/ -1., 1., 1. /)
WRITE (*,60) vektor_1, vektor_2, vektor_1*vektor_2
10 FORMAT (' Atama işlemleri ',/, &
  ' Vektör_1 = ', 3F8.2,/,' İndisli  = ', 3F8.2)
20 FORMAT (/' Toplama ve Çıkarma İşlemleri ',/, &
  ' Vektör_1 =      ', 3F8.2,/, ' Vektör_2 =      ', 3F8.2,/, &
  ' Vektör_1+vektor_2 = ', 3F8.2,/,' Vektör_1-Vektör_2 = ', 3F8.2)
30 FORMAT (/' Skalar sabit ile çarpım işlemi ',/, &
  ' Vektör_1 =      ', 3F8.2,/,' 2.*Vektör_1 = ', 3F8.2,/, &
  ' Vektör_1 * 2. = ', 3F8.2,/,' 2 *Vektör_1 = ', 3F8.2,/, &
  ' Vektör_1 * 2  = ', 3F8.2)
40 FORMAT (/' Skalar sabite bölme işlemi ',/, &
  ' Vektör_1 = ',3x, 3F8.2,/,' vektör_1 /5. =', 3F8.2,/, &
  ' Vektör_1 /5  =', 3F8.2)
50 FORMAT (/' Skalar çarpım işlemi ',/, &
  ' Vektör_1 = ', 3F8.2,/,' Vektör_2 = ', 3F8.2,/, &
  ' Vektör_1.SCALAR.Vektör_2 = ', 3F8.2)
60 FORMAT (/' Vektörel Çarpım işlemi ',/, &
  ' Vektör_1 = ', 3F8.2,/,' Vektör_2 = ', 3F8.2,/, &
  ' Vektör_1 X Vektör_2 = ', 3F8.2)
END PROGRAM Ornek5
```

Programının çıktısı

```
Atama işlemleri
Vektör_1 =     1.00    2.00    3.00
İndisli  =     1.00    2.00    3.00

Toplama ve Çıkarma İşlemleri
Vektör_1 =         10.00   20.00   30.00
Vektör_2 =          1.00    2.00    3.00
Vektör_1 + vektor_2 =    11.00   22.00   33.00
Vektör_1 - Vektör_2 =     9.00   18.00   27.00

Skalar sabit ile çarpım işlemi
Vektör_1 =          1.00    2.00    3.00
2. * Vektör_1 =     2.00    4.00    6.00
Vektör_1 * 2. =     2.00    4.00    6.00
2  * Vektör_1 =     2.00    4.00    6.00
Vektör_1 * 2  =     2.00    4.00    6.00

Skalar sabite bölme işlemi
Vektör_1 =       10.00   20.00   30.00
vektör_1 /5. =    2.00    4.00    6.00
Vektör_1 /5  =    2.00    4.00    6.00

Skalar çarpım işlemi
Vektör_1 =     1.00    2.00    3.00
Vektör_2 =     1.00    2.00    3.00
Vektör_1.SCALAR.Vektör_2 =    14.00

Vektörel Çarpım işlemi
Vektör_1 =     1.00   -1.00    1.00
Vektör_2 =    -1.00    1.00    1.00
Vektör_1 X Vektör_2 =    -2.00   -2.00    0.00
```

10.4 BİR MODÜLÜN İÇERİĞİNE ERİŞİMİ KISITLAMA

Bir modüle USE deyimi kullanılarak erişildiğinde, otomatik olarak modül içindeki tüm tanımlamalara erişilir. Modül'deki programların tamamına değil de sadece birkaçına erişimine istendiği durumlar söz konusu olabilir. Bu işleme *veri saklama* adı verilir.

Modüldeki bilgilerin programlar tarafından paylaşımını `PUBLIC` ve `PRIVATE` tanımlayıcıları ile sağlanır. Modül'deki bir unsur için `PUBLIC` tanımlaması yapılmışsa, bu unsura modül dışındaki programlardan erişilebilir. Modül'deki unsur `PRIVATE` tanımlaması yapılmışsa, bu unsura modül dışındaki programlar tarafından erişilemez ancak modül içinde tanımlanan alt programlar tarafından erişilebilir. Aksi belirtilmediği takdirde, modülde tanımlanan sabit, değişken ve alt programların tanımları `PUBLIC`'tir. Genel kullanım şekli

```
PUBLIC  :: sabit, değişken veya alt programlar
PRIVATE :: sabit, değişken veya alt programlar
```

Türetilmi veri tiplerinde modülde `PUBLIC` ve `PRIVATE` tanımlayıcıları kullanımı kuralları:

1. Bir modülde tanımlanan türetilmiş veri tiplerinin bileşenleri `PRIVATE` deyiminin kullanılmasıyla modül dışındaki kullanımı kısıtlanabilir. Örneğin,

```
TYPE vektor
     PRIVATE
     REAL :: x
     REAL :: y
END TYPE vektor
```

   Vektörün x ve y bileşenlerine erişim kısıtlanmış olmaktadır.
2. Önceki durumun tersine türetilmiş veri tipinin tamamı `PRIVATE` deyimi ile tanımlanabilir.

```
TYPE, PRIVATE :: vektor
     REAL :: x
     REAL :: y
END TYPE vektor
```

   Bu durumda da `vektor`'e modulü kullanan programlar tarafından erişilemez.
3. Türetilmiş veri tipi değişkenlerinin kullanımı, TYPE tanımlaması PUBLIC olmasına rağmen, sınırlandırılabilir. Örneğin

```
TYPE :: vektor
     REAL :: x
     REAL :: y
END TYPE vektor
TYPE (Vektor), PRIVATE :: vektor1, vektor2
```

   Bu durumda sadece `vektor1` ve `vektor2`'nin modül dışındaki kullanımları kısıtlanmıştır.

10.5 DİNAMİK BELLEK ALANI TAHSİS ETMEK

Bu bölüme kadar verilen örneklerde, indisli değişkenlerin tanımlama kısmında indisli değişkenin boyutu da tanımlamak suretiyle bellek alanı ayrılmıştı. Bu tür bellek alanı oluşturmaya *statik bellek tahsisi* adı verilir ve indisli değişkenlerin bellek alanları derleme ile oluşturulur ve bu alan asla değişmez. İndisli değişkenin boyutu (veya teknik anlamda bellek alanı) problemi çözebilmek için yeterince büyük olmalıdır. Örneğin, $n \times n$ 'lik bir lineer denklem sistemini çözmek için yazılacak programda *en fazla* kaç bilinmeyenli denklem sistemi çözülmek isteniyorsa, örneğin 1000 ise, programdaki matris ve vektörlerin boyutu minimum 1000 olmalıdır. Böylece problemi 5 bilinmeyenli 5 denklemden oluşan bir denklem sistemini çözmeye kalkıştığımızda, normalde gerekli bellek alanı çok küçük olmasına rağmen, bellek alanının %99'u boşu boşuna işgal edilmiş olur. Ayrıca oluşturulan çalışabilir (executable) programı, bellek yetersizliği nedeniyle, her bilgisayarda çalıştırılamayabilir. Diğer taraftan, indisli değişken boyutları küçük tutulursa, çözülmesi arzulanan boyutlardaki problemleri çözmede yetersiz kalabilir.
Bu problemi çözmenin birkaç yolu mevcuttur. Örneğin, programcı sadece kendisinin kullanacağı bir program ise, programı farklı boyutlarda derleyerek, çeşitli versiyonlarını

oluşturabilir. Ancak ticari olarak satılacak veya farklı şehirlerde farklı birimlerde farklı bilgisayar sistemlerinde çalıştırılacak (bankacılık sektörü gibi) bir program hazırlanıyorsa, bu durumda her durum ve koşula uygun bir program hazırlanmalıdır. Bu durumda en iyi çözüm *dinamik bellek alanı tahsis etme* yöntemidir. Bu yolla, program çalıştırıldığında bellek alanında mevcut yere göre indisli değişkenlerin boyutlarını ayarlar. Bu şekilde hem bellek alanı boşu boşuna işgal edilmez, hem de küçük veya büyük bellek alanına sahip bilgisayarlarda çalıştırılabilir.

Dinamik bellek kullanan Fortran indisli değişkenleri tanımlama kısmında `ALLOCATABLE` niteliği ile tanımlanır ve bellek alanı `ALLOCATE` deyimi ile oluşturulur. Programın bellek alanı ile işi bittiğinde, bu bellek alanı `DEALLOCATE` deyimi ile boşaltılır. `ALLOCATABLE` niteliği ile indisli değişken tanımlamanın genel kullanım şekli

```
Tip, ALLOCATABLE, DIMENSION(:,:) :: indisli
```

olarak verilmektedir. Burada indisli değişken `REAL`, `INTEGER`, `REAL*8` vb `Tip`lerde olabilir. İndisli değişken, bir, iki veya daha fazla indisli olabilir, ancak her indisin aralığı veya maksimum boyutu iki nokta üst üste (`:`) ile girilir çünkü indisli değişkenin üst indis sınırının ne olabileceği bilgisayar tarafından belirlenecektir. İki nokta üst üste ile indisleri tanımlanan değişkenlere *ertelenmiş-şekilli indisli değişken* adı verilir çünkü indisin üst sınırı bellek alanı tahsis edilinceye kadar belirsizdir.

Program çalıştırıldığında, indisli değişkenin gerçek uzunluğu (veya ebadı) `ALLOCATE` deyimi ile belirlenecektir. `ALLOCATE` deyiminin genel kullanım şekli

```
ALLOCALTE(bellek alanı tahsis edilecek indisli değişkenler, STAT=statü)
```

olarak verilmektedir. `STAT` tamsayı değeri alır tercihe bağlı olarak kullanılabilir; 0 değeri aldığında bellek tahsisi “başarılı” pozitif bir tamsayı değer aldığında bellek tahsisi “başarısız” olduğunu belirtir. Örneğin,

```
ALLOCATE ( abc(500,0:99),STAT=statu)
```

deyimi ile program çalıştırıldığında `abc` indisli değişkenine 500×100’lik bellek alanı tahsis edilmektedir.

Bellek alanı tahsisi, hemen değil de, indisli değişken kullanılıncaya kadar geciktirilebilir. Henüz dinamik bellek tahsisi yapılmamış indisli değişken kullanımına teşebbüs edilirse, program “çalıştırma hatası” (run-time error) verir ve “başarısız” çıkış ile son bulur. Fortran 90/5’in mantıksal arşiv fonksiyonlarından `ALLOCATED()` bir indisli değişken kullanılmadan önce bellek tahsisi statüsünü test etmede kullanılabilir.

Örneğin, aşağıda verilen program parçasında `abc` indisli değişkeninin statüsünü kullanmadan önce test etmektedir.

```
REAL, ALLOCATABLE, DIMENSION(:) :: abc
...
```

```
IF( ALLOCATED(abc) ) THEN  ! .TRUE. durumu
     READ(birim,*) abc
ELSE                       ! .FALSE. durumu
     WRITE(*,*)' !!!!! UYARI !!!!!!'
     WRITE(*,*)' Bellek alanı tahsis edilmedi'
END IF
```

Program veya alt programda, bellek alanı tahsis edilen değişken kullanıldıktan (tekrar kullanılmayacaksa) sonra tahsis edilen bellek alanını DEALLOCATE deyimi ile boşaltmak gerekir. DEALLOCATE deyiminin genel kullanım şekli

DEALLOCALTE(bellek alanı boşaltılacak indisli değişkenler, **STAT**=statü)

olarak verilir. Bellek alanı tahsisi yapılmış değişkene artık programınızda ihtiyacınız yoksa bellek alanını boşaltmak, programı hızlandıracaktır.

ÖRNEK 6: Tek indisli gerçek sayılardan oluşan, bellek sahasına göre alan tahsisi yapılan bir indisli değişken oluşturunuz. Bu indisli değişkenin elemanları toplamını hesaplayınız.

```
PROGRAM Ornek6
IMPLICIT NONE
REAL , ALLOCATABLE, DIMENSION(:) :: x
INTEGER :: Boyut, i
REAL :: Topla
READ*, Boyut
ALLOCATE(X(1:Boyut))
DO i=1,Boyut
   X(i)=i           ! Başlangıç değerleri indis değerine atayalım
END DO
Topla=0.0
DO i=1,Boyut
   Topla=Topla+x(i)
END DO
DO i=1,Boyut
   PRINT*, x(i)     ! Değerleri ekrana yaz
END DO
DEALLOCATE(x)       ! x  ile işimiz bitti. Belleği boşalttık
PRINT*, 'Toplam=',Topla
END PROGRAM Ornek6
```

10.6 GÖSTERGE (POINTER) KULLANIMI

Fortran 90/95 dilinde bir başka değişken tipi daha tanımlanabilir ki bu değişken hiçbir suretle veri içermez; sadece bir başka değişkenin *bellek adresini* içerir. Bu tür değişkenlere *gösterge* (*pointer*) adı verilir.

Göstergeler, bir programın çalışması esnasında, skalar veya indisli değişkenlerin dinamik

olarak oluşturulması veya imhasında, kaç değişkenin bir işleme maruz kalacağı bilinmediği durumlarda kullanılır.

10.6.1 GÖSTERGE VE HEDEF'LER

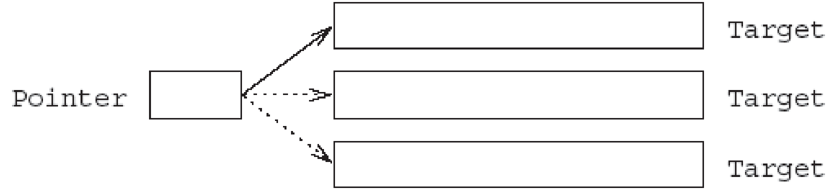
Gösterge olarak kullanılacak değişkenler, tip tanımlama kısmında POINTER niteliği ile beraber tanımlanmalıdır.

```
REAL, POINTER  :: g1
TYPE (vektor), POINTER :: vektor_gosterge
INTEGER, DIMENSION(:), POINTER :: gost1
```

Bir gösterge, göstergenin tipinde herhangi bir skalar veya indisli değişken *hedef* olarak tanımlandığı sürece işlev görür. *Hedef*, gösterge ile kullanılmaya müsait, adresleri bilinen verilerdir. Hedef olarak tanımlanan değişkenler **TARGET** niteliği ile beraber tanımlanır. Örneğin,

```
INTEGER, TARGET :: a, b, c
REAL, TARGET :: u, v
INTEGER, DIMENSION(49), TARGET :: indisli
```

Bir gösterge, aynı tipte tanımlanmış birden fazla hedef için kullanılabilir.



Hedef değişkenin tipi, türü (tek/çift hassasiyet) ve/veya rankı tanımlama satırı ile belirlenir. İndisli gösterge boyutları geciktirilmiş boyutlama ile verilmelidir. Hedef indisli değişkenin rankı sabit olmasına karşın şekli değişebilir.

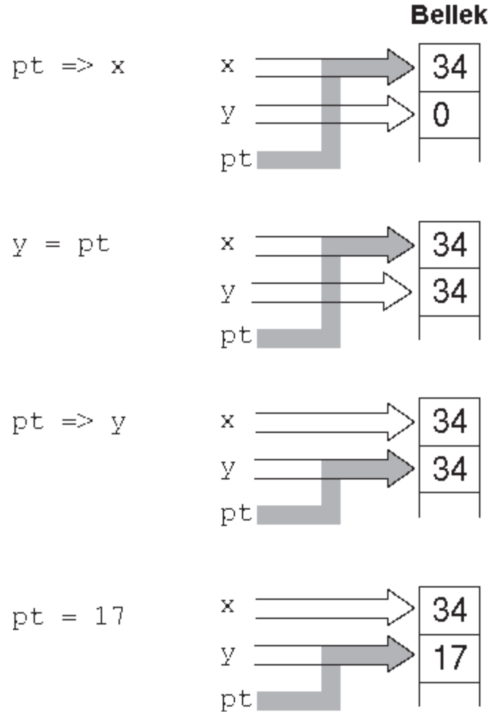
10.6.2 GÖSTERGE ATAMA DEYİMİ

Gösterge atama deyiminin kullanımı genel olarak aşağıdaki şekilde verilmektedir.

```
Gösterge (Pointer) => Hedef (Target)
```

Burada `Gösterge` gösterge değişkeninin ismi, `Hedef` gösterge ile aynı tipte olan hedef değişkeninin ismidir. Atama işlemi `=>` karakterlerinin boşluksuz bir şekilde yan yana kullanılması ile gerçekleştirilir.

Örneğin, aşağıdaki programı ele alalım.




```
INTEGER, POINTER :: pt
INTEGER, TARGET :: x=34, y=0
!
pt => x  ! pt, x'i gösteriyor
y = pt   ! y eşit x
pt => y  ! pt, y'i gösteriyor
pt = 17  ! y eşit 17
```

Bu örnekten görüldüğü üzere gösterge bellek alanına sahip değil sadece bellek adreslerini bilmekte ve bu bellek adreslerindeki değerleri almaktadır. Bir başka örnek de

```
REAL, POINTER :: g
REAL, TARGET  :: t1=-5, t2=5
g=>t1
PRINT*, g, t1, t2
g=>t2
PRINT*, g, t1, t2
```

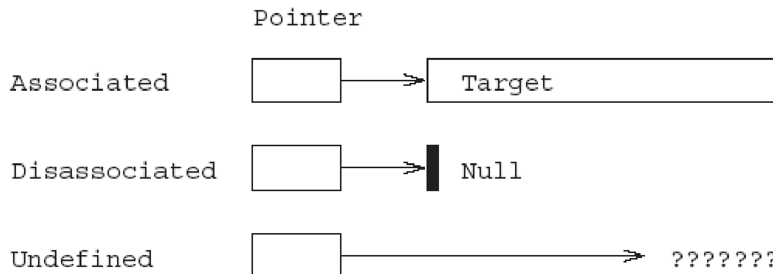
Programının çıktısı

```
     -5.000000     -5.000000      5.000000
      5.000000     -5.000000      5.000000
```

şeklinde gerçekleşir.

10.6.3. GÖSTERGE STATÜSÜ

Gösterge durumu veya statüsü, göstergenin geçerli bir hedefi gösterip göstermediğini belirtir. Bir gösterge değişkeninin üç durumda olabilir: *belirsiz* (undefined), *tanımlı* (associated) ve *tanımsız* (disassociated). Bir gösterge, tip tanımlama satırında ilk tanımlamada gösterge statüsü *belirsiz*' dir. Gösterge bir hedef ile => ataması ile ilişkilendirildiğinde, statüsü *tanımlı* hale gelir. Daha sonra göstergenin hedef ile olan ilişkisi sonlandırıldığında, yeni bir hedef ile ilişkilendirilmemiş ise, gösterge statüsü *tanımsız*dır. Aşağıdaki diyagram göstergenin şematik olarak statü seçeneklerini belirtmektedir:



Bir göstergenin, hedef ile olan ilişkisi nasıl kesilir? yani hedeften nasıl ayrılır? Bu işlem

göstergenin bir başka hedefe atama yapılması ile gerçekleştirilebilir; ayrıca bir gösterge `NULLIFY` deyimi ile tüm hedeflerle olan ilişkisi kesilebilir (tanımlı durumdan tanımsız duruma geçmek). `NULLIFY` deyiminin genel kullanım şekli aşağıda verilmektedir:

```
NULLIFY (gösterge_1, [, gösterge_2,...])
```

Burada *gösterge_1* ve *gösterge_2* vs gösterge değişkenleridir. Bu deyimin icra edilmesinden sonra, deyim ile verilen tüm göstergelerin hedefleriyle ilişikleri kesilir.

Bir gösterge tanımlı (ilişkili) olmadığı bir hedefi olmadan kullanılamaz. Bu nedenle, herhangi bir göstergenin belirli bir hedefle tanımlandığını bilmemiz gerekir. Bir mantıksal fonksiyon olan `ASSOCIATED` bu ilişkiyi belirlemek amacıyla kullanılan bir arşiv fonksiyonudur. Bu fonksiyon iki şekilde kullanılabilir: birinci kullanım şekli

```
statü = ASSOCIATED(gösterge)
```

olarak verilmektedir. Eğer `gösterge` bir hedef ile tanımlı veya ilişkili ise `true`, ilişkili değilse `false` değerini getirir. İkinci kullanım şekli

```
statü = ASSOCIATED(gösterge, hedef)
```

olarak verilmektedir. Eğer `gösterge` bir `hedef` ile tanımlı veya ilişkili ise `true`, değilse `false` değerini getirir.

Bir göstergenin ilişkili statüsü, göstergenin tanımlandığı andan ilk kullanımına kadar *belirsiz*dir. Bundan sonra, göstergenin statüsü ya tanımlı yada tanımsızdır. Örneğin, bir programda göstergeler aşağıdaki gibi tanımlanır veya geçersiz kılınır:

```
REAL, POINTER :: gosterge1, gosterge2
INTEGER, POINTER :: t1
     ...
( başka deyimler )
     ...
NULLIFY (gosterge1, gosterge2, t1)
```

Aşağıdaki program `ASSOCIATED` ve `NULLIFY` arşiv fonksiyonlarının kullanımını belirtmektedir.

```
PROGRAM gosterge
IMPLICIT NONE
INTEGER, POINTER :: g1, g2, g3
INTEGER, TARGET :: a = 50, b = 25, c = 9
NULLIFY ( g1, g2, g3)     ! Nullify pointers
```

```
WRITE (*,*) ASSOCIATED(g1)
g1 => a        ! g1, a'yı gösteriyor
g2 => b        ! g2, b'yi gösteriyor
g3 => c        ! g3, c'yi gösteriyor
WRITE (*,*) ASSOCIATED(g1)
WRITE (*,*) ASSOCIATED(g1,b)
END PROGRAM gosterge
```

Program çalışmaya başlar başlamaz `g1`, `g2` ve `g3` göstergeleri geçersiz (tanımsız) kılınır; bu nedenle, `ASSOCIATED(g1)`, sonucu `false`'tır. Sonra `g1`, `g2`, `g3` göstergeleri `a`, `b` ve `c` hedefleri ile tanımlanıyorlar, bu nedenle `ASSOCIATED(g1)` fonksiyonunun sonucu `true` olur. Ancak `ASSOCIATED(g1,b)` fonksiyonunun sonucu `false` olur çünkü `g1`, `b` ile tanımlı değildir.

10.6.4 İNDİSLİ DEĞİŞKENLERDE GÖSTERGE KULLANIMI

Bir gösterge skalar bir değişkeni gösterebileceği gibi indisli değişkeni de gösterebilir. Bir gösterge tip tanımlamada ilişkilendirileceği indisli değişkenin boyutu ile tanımlanmalıdır. Örneğin,

```
REAL, DIMENSION(40,60), TARGET :: veri
REAL, DIMENSION(:,:), POINTER  :: gostergem
gostergem => veri
```

Bir gösterge sadece bir indisli değişkeni değil aynı zamanda indisli değişkenin alt kümelerini de gösterebilir. Örneğin, aşağıdaki programda `bilgi` indisli değişkeni `1`'den `16`'a kadar tamsayı değerleri almaktadır. `gos1` tüm indisli değişkeni göstermektedir. `gos2(2::2)` orijinal indisli değişkenin çift indisli (2, 4, 6,…) değerlerini gösterir. `gos3(2::2)` de `gos2` indisli değişkenin çift indisli (4, 8,…) değerlerini gösterir ve işlem bu şekilde devam eder.

```
PROGRAM indis_gosterge
IMPLICIT NONE
INTEGER :: i
INTEGER, DIMENSION(16), TARGET :: bilgi
INTEGER, DIMENSION(:), POINTER :: gos1, &
      gos2, gos3, gos4, gos5
bilgi = (/ (i, i=1,16) /)
gos1 => bilgi
gos2 => gos1(2::2)
gos3 => gos2(2::2)
gos4 => gos3(2::2)
gos5 => gos4(2::2)
WRITE (*,'(A,16i3)') ' gos1=', gos1
WRITE (*,'(A,16i3)') ' gos2=', gos2
WRITE (*,'(A,16i3)') ' gos3=', gos3
WRITE (*,'(A,16i3)') ' gos4=', gos4
WRITE (*,'(A,16i3)') ' gos5=', gos5
END PROGRAM indis_gosterge
```

Çıktı

```
gos1=  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
gos2=  2  4  6  8 10 12 14 16
gos3=  4  8 12 16
gos4=  8 16
gos5= 16
```

şeklinde elde edilir.

ALIŞTIRMALAR

10.1 Aşağıdaki program parçasında varsa hataları tespit ediniz

```
TYPE konum_vektoru
  LOGICAL  :: Ok
  REAL     :: x
  REAL     :: y
  REAL     :: zaman
  CHARACTER(LEN=5) :: mevki
END TYPE konum_vektoru
TYPE(konum_vektoru), DIMENSION(99):: yorunge
```

10.2 Türetilmiş veri tipi kullanarak `kisi` isimli veri tipi oluşturunuz. Kişisel bilgiler (1) adı (`LEN=12`), (2) soyadı (`LEN=12`), (3) yaşı (`INTEGER`), (4) doğduğu il (`LEN=10`), (5) aylık geliri (`REAL`)'den oluşmakta ve `BILGI.DAT` isimli bir kütükten formatsız olarak okunmaktadır. Yazacağınız bir program ile veri tabanında yer alan kişilerin (a) geliri maksimum olandan minimum olana, (b) en yaşlıdan en gence, (c) doğduğu ile göre alfabetik olarak sıralama yapan bir program yazınız.

10.3 A(x_0, y_0) ile B(x_1, y_1) noktalarından geçen bir doğrunun eğimini veren, noktaları türetilmiş veri tipi olarak tanımlayıp kullanarak, bir program yazınız.

10.4 `kutupsal` isimli bir türetilmiş veri tipi [$(r,\theta)= re^{i\theta} = r\cos\theta + ir\sin\theta$] tanımlayınız. Programa girilen (r_1,θ_1) ve (r_2,θ_2) (yani $z_1 = r_1e^{i\theta_1}$ ve $z_2 = r_2e^{i\theta_2}$ karmaşık sayıları) için iki karmaşık sayının çarpımını, bölümünü ve *n*.ci dereceden kuvvetini $z_1z_2 = r_1r_2e^{i(\theta_1+\theta_2)}$, $z_1 / z_2 = r_1 / r_2e^{i(\theta_1-\theta_2)}$ ve $z_1^n = r_1^n e^{in\theta}$ formülleriyle hesaplayan bir program yazınız.

10.5 Uzunluk birimleri (cm, m, inç, feet, mil, mikron, Angtröm) arasında birim dönüştürme işlemi yapan bir program hazırlamak istiyoruz. Program herhangi bir birimde girilen uzunluk birimini diğer birimler cinsinden ifade etsin istiyoruz. Programınızda aşağıdaki gibi `uzunluk` isimli türetilmiş veri tipi kullanın. Dönüştürme katsayıları aşağıda verilmiştir.

```
TYPE uzunluk
     Cm        :: REAL
     M         :: REAL
     Inc       :: REAL
     Mil       :: REAL
     Mikron    :: REAL
     Angtrom   :: REAL
END TYPE uzunluk
```

Cm	m	inç	feet	Mil	Mikron	Angtröm
100	1	39.37	3.281	6.214×10^{-4}	10^6	10^{10}

10.6 Yukarıda verilen soruyu hacim (cm^3, m^3, lt, inç^3, ft^3, yard^3, galon) birimleri için tekrarlayınız.

10.7 Tarih isimli bir veri tipi oluşturarak, girilen tarihi `Gun`, `Ay`, `YIL` olarak okuyan ve yazan bir program yazınız.

10.8 Aşağıdaki programların çıktısını bulunuz.

(a)
```
PROGRAM p8a
INTEGER , POINTER :: A,B
INTEGER , TARGET :: C
INTEGER :: D
C = 1
A => C
C = 2
B => C
D = A + B
PRINT *, A, B, C, D
END PROGRAM p8a
```

(b)
```
PROGRAM p8b
IMPLICIT NONE
REAL , TARGET :: T1 = 21
REAL , TARGET :: T2 = 30
REAL , POINTER :: g1, g2
g1 => T1
g2 => T2
PRINT *, ' g1= ', g1, ' g2= ', g2
g1=g2
PRINT *, ' g1= ', g1, ' g2= ', g2
PRINT *, ' T1= ', T1, ' T2= ', T2
END PROGRAM p8b
```

(c)
```
PROGRAM p8c
IMPLICIT NONE
REAL , DIMENSION(:) , POINTER :: X
REAL , DIMENSION(1:10) , TARGET :: Y
INTEGER , PARAMETER :: Boyut=10000
INTEGER :: i
ALLOCATE(X(1:Boyut))
DO i=1,Boyut; X(i)=i; END DO
DO i=1,10; PRINT*, X(i); END DO
DO i=1,10; Y(i)=i*i; END DO
DO i=1,10; PRINT*, Y(i); END DO
X=>Y
DO i=1,10; PRINT*, X(i); END DO
END PROGRAM p8c
```

(d)
```
PROGRAM p8d
TYPE Link
```

```
          CHARACTER :: C
          TYPE(Link), POINTER :: Sonraki
     END TYPE Link

     TYPE (Link) , POINTER :: Kok , Guncel
     INTEGER :: IO_Statusu=0
          ALLOCATE(Kok)
          READ (UNIT=*, FMT='(A)', ADVANCE='NO', &
                IOSTAT=IO_Statusu) Kok%C
          IF(IO_Statusu<=0) THEN
               NULLIFY(Kok%Sonraki)
          ELSE
               ALLOCATE(Kok%Sonraki)
          ENDIF
          Guncel => Kok
          DO WHILE (ASSOCIATED(Guncel%Sonraki))
               Guncel => Guncel%Sonraki
               READ(UNIT=*,FMT='(A)',ADVANCE='NO',&
                    IOSTAT=IO_Statusu) Guncel%C
               IF(IO_Statusu<=0) THEN
                    NULLIFY(Guncel%Sonraki)
               ELSE
                    ALLOCATE(Guncel%Sonraki)
               ENDIF
          END DO
          Guncel => Kok
          DO WHILE (ASSOCIATED(Guncel%Sonraki))
               PRINT * , Guncel%C
               Guncel => Guncel%Sonraki
          END DO
     END PROGRAM p8d
     ! Bu program girdileri
     !  ab  de  bos
     !  cd  fg  uc
     !  ne  and

     PROGRAM p8e
     IMPLICIT NONE
     REAL , POINTER :: P1 => NULL()
     REAL , POINTER :: P2 => NULL()
       ALLOCATE(P1)
       P1=21.0
       P2=>P1
       PRINT *,P1
       PRINT *,P2
       P2=P2+1
       PRINT *,P1
       PRINT *,P2
       DEALLOCATE(P1)
       PRINT *,P2
     END PROGRAM p8e
```

BÖLÜM 11
SERİLER VE TÜREV

11.1 SONLU VE SONSUZ SERİLER

Seriler sonlu veya sonsuz toplam içeren, bazen de sonlu çarpım v.s şeklinde ifade edilebilen aritmetik işlemlerdir. Örneğin,

$$\sum_{n=1}^{10}\frac{1}{1+n^2} \quad \text{ve} \quad \sum_{n=1}^{\infty}\frac{n+1}{\sqrt{n}+n^2} \quad \text{veya} \quad \prod_{n=1}^{20}\frac{1+n}{1+n^2}$$

ifadeleri sonlu ve sonsuz serilere tipik örneklerdir. Bu serilerin toplamı, bir sayaç ve `DO-END DO` döngüsü aracılığıyla hesaplanabilir. Öte yandan, üst sınırı sonsuz olan serilerde, seri toplam arzu edilen basamak hassasiyetine göre kaç teriminin toplanması gerektiğini bilmemiz gerekir.

Matematik derslerinizden hatırlayacağınız üzere serilerde bir *yakınsaklık* kavramı vardır. Bu nedenle, serilere bir takım yakınsaklık kriterleri (kıyaslama, orantı, kök ve integral testi gibi) uygulayarak, serinin yakınsak olup olmadığını araştırmayı öğrendiniz. Hatırlarsanız, yakınsak bir serinin yeterli sayıda terimini topladıktan sonra, diğer terimlerin de ilave edilmesiyle toplamın değişmediğine ya da çok az değiştiğine şahit oldunuz. Bunun nedeni, yakınsak serilerde ilave edilen her yeni terimin bir önceki terimden daha küçük olmasıdır. Böylece ilave edilen terim sayısı arttıkça, ilave edilen yeni terim sıfıra yaklaşacaktır. Bu durumda, toplama işlemini ilave edilen terimin sıfıra yaklaştığı, belirli bir basamak hassasiyetinin altında kalan bir yerden kesmek ve diğerler terimleri ihmal etmek durumundayız.

Burada en önemli problem, hangi değeri sıfıra yakın kabul edeceğimizdir. Bu problem, tamamen serinin değerini hesaplamak isteyen kişinin, o seri toplamını kaç basamağa kadar hassas bir şekilde hesap etmek istemesine bağlıdır. Bir kural olarak belirtmemiz gerekirse, üç basamağa kadar hassas olarak hesaplanmak istenen bir seri toplamına ilave edilecek bir sonraki terimin 0.0001, dört basamak hassasiyet için 0.00001'den küçük olması yeterlidir. Bu nedenle, toplama işlemini yaparken her defasında ilave edilecek terimi ayrı bir yerde hesaplayıp, bunun mutlak değerinin arzu edilen epsilon değerinden (bunu ε ile belirtiriz ve bu değer, tek hassasiyetli işlem yaparken, 10^{-6}'dan, çift hassasiyetli işlem yaparken de 10^{-14}'den daha küçük seçilmemelidir) küçük olup olmadığı kontrol edilir. Eğer ilave edilecek yeni terim bu ε değerinden küçük ise, toplama işlemi sona erdirilir. Iraksayan bir seride ilave edilen her yeni terim bir öncekine göre büyür veya serinin yeni değeri sıfırdan farklı bir değerde sabitlenir ki , bu durumda, serinin toplamının sonsuz olmasına neden olur; sonsuz döngüye girer.

Bir sayısal örnek üzerinde açıklayalım. Yakınsadığını bildiğimiz bir seri aşağıdaki şekilde verilmiştir:

$$\begin{aligned}\sum_{n=1}^{\infty}\frac{1}{n^4} &= 1+\frac{1}{2^4}+\frac{1}{3^4}+\cdots+\frac{1}{10^4}+\frac{1}{11^4}+\cdots \\ &= 1+0.0625+0.0123456+\cdots+0.0001+0.00000683+\cdots\end{aligned}$$

Bu örnekten de görüleceği üzere, serinin terimlerinin soldan sağa doğru gidildikçe, bir önceki terime oranla küçüldüğü ve belirli bir noktadan sonra sıfıra yaklaştığı gözlenmektedir. Eğer bu seri toplamını üç basamağa kadar hassas bir şekilde hesaplamak istiyorsak, bu durumda seri toplamının ilk 10 terimini toplamak yeterli olacaktır; çünkü değeri 0.0001 olan 10.cu terimden, yani, $1/10^4$'den, sonraki terim 0.0001'den daha küçüktür ve 11, 12, ... terimlerinin ilavesiyle seri toplamın 3.cü değil, sadece 4 veya 5.ci basamağı değişecektir.

Şimdi bu serinin programlanmasını ele alalım. Bir program aşağıdaki şekilde hazırlanabilir:

```
PROGRAM sonsuz_seri
IMPLICIT NONE
INTEGER :: n
REAL :: nn, terim, Seri_toplam
Seri_toplam=0.0
n=0
DO
   n=n+1          ! Terim sayacı
   nn=REAL(n)
   terim=1.0/nn**4
   IF(ABS(terim)<EPSILON(1.)) EXIT
   Seri_toplam=Seri_toplam + terim
END DO
PRINT*, 'Serinin ilk',n,' terimi toplandı'
PRINT*, 'TOPLAM = ',Seri_toplam
END PROGRAM Sonsuz_seri
```

Programının çıktısı

```
Serinin ilk 54  terimi toplandı
TOPLAM =   1.08232105
```

Bu programda sayaç (`n`) ve toplam (`Seri_toplam`) sıfırlandıktan sonra, sayaç bir artırılır. Serinin ilk terimi ($a_1 = 1/1^4$) hesaplanır; mutlak değerce (çünkü seri sıfıra pozitif veya negatif taraftan yakınsayabilir) ε değerinden küçük olup olmadığına bakılır. Terimin ilk aşamalarda bu kriteri sağlamayacağı aşikardır. Bu kıyaslamayı sağlamadığında, terim toplama ilave edilir (`Seri_toplam`←`Seri_toplam+terim`) ve sayaç bir artırılarak aynı işlemlere, $|\text{TERIM}| < \varepsilon$ şartı sağlanıncaya kadar devam ettirilir. Şartın sağlandığı durumdaki `Seri_toplam` değeri tek basamak makine hassasiyetine göre doğru olarak hesaplanmış seri toplamını verecektir. Yukarıdaki programda arşiv fonksiyonlarından `EPSILON(1.)` kullanılmıştır. Programın çalıştırıldığı bilgisayarda `EPSILON(1.)=1.1921E-7` olup, seriyi 6 basamak hassasiyetle toplama işleminin gerçekleştirildiği anlamına gelir. Seri toplamını, örneğin 4 basamak hassasiyetle yapmak istersek yukarıdaki programda **`EPSILON(1.)`** yerine **`1.E-5`** girmemiz yeterli olacaktır. Bu durumda program çıktısı

```
Serinin ilk 18  terimi toplandı
TOPLAM =   1.08226109
```

olarak elde edilir.

Diğer taraftan, sonlu serilerin toplamında yakınsama ya da yakınsamama gibi bir problemimiz yoktur. Bu serileri programlarken iki yol izlenebilir. Bunlardan birisi `DO-WHILE`, diğeri de indisli `DO` döngüsü kullanmayı içermektedir. Bu iki yolu açıklamak için aşağıda verilen sonlu serinin toplamına bir göz atalım.

$$\sum_{i=1}^{13} \frac{i+3}{i^2+3i-1}$$

`DO-WHILE` döngüsü kullanımı ile programlamaya örnek olarak

```
PROGRAM sonlu_seri1
IMPLICIT NONE
REAL    :: i, pay, payda, Seri_toplam=0.0
INTEGER :: n=0     ! sayaç değişkeni
! Sonlu Seri Toplam
DO
   n=n+1           ! sayaç kullanılır
   i=REAL(n)
   pay  = i + 3.0
   payda= i*i + 3.0*i - 1.0
   Seri_toplam=seri_toplam + pay/payda
   IF(n=13) EXIT
END DO
PRINT*,'SERI TOPLAM = ', Seri_toplam
END PROGRAM Sonlu_seri1
```

programı verilebilir. Burada yukarıda verilen sonsuz serinin programlanması için benimsenen algoritma aynen kullanılmıştır; ancak toplam ilk 13 terimi içermesi gerektiğinden, toplama işleminin sona erdirilmesi için uygulanan kriter, terimin ε gibi bir sayıdan küçük olmasından ziyade, sayacın değerinin 13 olmasına bağlanmıştır.

Bir indisli `DO` döngüsü kullanarak programlanması durumunda, program

```
PROGRAM sonlu_seri2
IMPLICIT NONE
REAL    :: i, pay, payda, Seri_toplam=0.0
INTEGER :: n
! Sonlu Seri Toplam
DO n=1,13
   i=REAL(n)
   pay  = i + 3.0
   payda= i*i + 3.0*i - 1.0
   Seri_toplam=seri_toplam + pay/payda
END DO
PRINT*,'SERI TOPLAM = ', Seri_toplam
END PROGRAM Sonlu_seri2
```

şeklinde de yazılabilir. Bu programda da toplama işlemi yapılan blok, bir `DO-END DO` döngüsü içine yerleştirilmiştir. Döngü bu bloğu 13 kez (`n`'nin 1-13 arasındaki değerleri için) işleyecektir. Döngü çıkışında `seri_toplam` sonlu seri toplamını verir.

`DO-END DO` yapısı esasen bir sayaç ve kıyaslamayı içermektedir. Örneğin,

```
DO n=i,j,ib
     ...
END DO
```

genel şekliyle verilen bir `DO` döngüsünde, bilgisayar bu deyim çiftini işlerken döngü sayacını, indisin ilk değerine atar (**n←i**), sonra `END DO` deyiminin bulunduğu satıra gelindiğinde, döngü indisinin üst sınırı olan **j** değerine ulaşıp ulaşmadığını kontrol eder. Bu değere ulaşmış veya bu değeri aşmış ise, döngü sona erer ve bilgisayar işlemek üzere `END DO`'dan sonraki satıra geçer. Fakat sayaç (**n**), indis üst değerine henüz ulaşmamış ise, bu durumda sayacı **ib** kadar artırır (**n←n+ib**).

ÖRNEK 1: Aşağıdaki şekilde verilen ve katsayıları programa dışarıdan temin edilen serinin toplamını veren bir program yazınız.

$$\sum_{i=1}^{7} a_i \sum_{j=0}^{9} b_{i,j}$$

Yukarıdaki seri toplamının açılımı aşağıdaki şekildedir:

$$a_1\ (b_{1,0} + b_{1,1} + b_{1,2} + b_{1,3} + b_{1,4} + b_{1,5} + b_{1,6} + b_{1,7} + b_{1,8} + b_{1,9})\ +$$
$$a_2\ (b_{2,0} + b_{2,1} + b_{2,2} + b_{2,3} + b_{2,4} + b_{2,5} + b_{2,6} + b_{2,7} + b_{2,8} + b_{2,9})\ +$$
$$a_3\ (b_{3,0} + b_{3,1} + b_{3,2} + b_{3,3} + b_{3,4} + b_{3,5} + b_{3,6} + b_{3,7} + b_{3,8} + b_{3,9})\ +$$
$$a_4\ (b_{4,0} + b_{4,1} + b_{4,2} + b_{4,3} + b_{4,4} + b_{4,5} + b_{4,6} + b_{4,7} + b_{4,8} + b_{4,9})\ +$$
$$a_5\ (b_{5,0} + b_{5,1} + b_{5,2} + b_{5,3} + b_{5,4} + b_{5,5} + b_{5,6} + b_{5,7} + b_{5,8} + b_{5,9})\ +$$
$$a_6\ (b_{6,0} + b_{6,1} + b_{6,2} + b_{6,3} + b_{6,4} + b_{6,5} + b_{6,6} + b_{6,7} + b_{6,8} + b_{6,9})\ +$$
$$a_7\ (b_{7,0} + b_{7,1} + b_{7,2} + b_{7,3} + b_{7,4} + b_{7,5} + b_{7,6} + b_{7,7} + b_{7,8} + b_{7,9})$$

Dikkat edilirse önce içerdeki `j` üzerinden toplamın hesaplanması gerekir; daha sonra `a(i)`'ler çarpılmalı ve bu çarpımlar dıştaki i üzerinden toplamı hesaplanmalıdır. Program a_i ve $b_{i,j}$ katsayıları ekrandan temin edilecek şekilde aşağıdaki gibi yazılabilir:

```
PROGRAM Ornek1
IMPLICIT NONE
REAL, DIMENSION(7) :: A
REAL, DIMENSION(1:7,0:9) :: B
REAL :: T1, T2
INTEGER :: i, j
DO i=1,7
   READ*, A(i)        ! a(i)'leri oku
END DO
DO i=1,7
   DO j=0,9
       READ*, B(i,j) ! b(i,j)'leri oku
   END DO
END DO
!  Seri toplam işlem bloğu
```

```
T1=0.0
DO i=1,7
   T2=0.0
    DO j=0,9
          T2=T2+B(i,j)
    END DO
   T1=T1+A(i)*T2
END DO
PRINT*, T1
END PROGRAM Ornek1
```

(İçteki döngü: $\sum_j$; dıştaki döngü: $\sum_i$)

Seri toplamı yapabilmemiz için önce içteki toplam, yani herhangi bir i için $\sum_j b_{i,j}$ toplamım hesaplamak gerekir. Bu nedenle `DO` döngüsünden önce bu toplamı verecek `T2` değişkeni sıfırlanmıştır. Döngü çıkışında toplam herhangi bir `i` değeri için hesaplanmış olur. Daha sonra dıştaki toplam, içteki işlemlerin, yani a_i ile `T2`'nin çarpımlarının toplamını verir. Bu toplam `T1` değişkeni ile temsil edilmiştir.

Programlama mantığını anlamak için verilen seri toplamların açılımını kafanızda tasarlayabilmelisiniz.

ÖRNEK 2: Aşağıdaki şekilde verilen seri çarpımını, dışarıdan girilen herhangi bir x değeri için hesaplayan bir program yazınız ve bulduğunuz sonucu $\cos x$ ile karşılaştırınız.

$$\prod_{n=1}^{10}\left(1-\frac{4x^2}{(2n-1)^2\pi^2}\right)=\left(1-\frac{4x^2}{\pi^2}\right)\left(1-\frac{4x^2}{9\pi^2}\right)\left(1-\frac{4x^2}{25\pi^2}\right)\cdots\left(1-\frac{4x^2}{19^2\pi^2}\right)$$

Bu serinin açılımını göz önüne alınarak, programı aşağıda verilmiştir:

```
PROGRAM Ornek2
IMPLICIT NONE
REAL :: x, carpim, x_kare, pi_kare
REAL :: pi=3.14159, oran, terim
INTEGER :: n, nn
PRINT*, ‘x değerini gir’; READ*, x  ! x değerini gir
pi_kare= pi*pi; x_kare =4.*x*x
oran   = x_kare/pi_kare
carpim=1.-oran              ! carpım ← 1 − 4x^2 / π^2
DO n=2,10
   nn    = REAL(2*n-1)
   Terim = oran/(nn*nn) !  terim ← (4x^2 / π^2) /(2n − 1)^2
   !  carpım ← carpim*[1 − (4x^2 / π^2) /(2n − 1)^2]  işlemini uygula
   carpim= carpim*(1.-terim)
END DO
PRINT*, 'Seri çarpım=',carpim,' cosx=',COS(x)
END PROGRAM Ornek2
```

Faktoriyel hesabında olduğu gibi, ilk terimi döngü dışında bırakarak çarpım değişkenine atıyoruz. Döngüyü ikinci terimden son terime kadar olan terimler için kuruyoruz. $\cos x$ ile karşılaştırdığımızda 0.5'den küçük x değerleri için sonuçların 2 veya 3 basamak aynı olduğunu gözleriz. Terim sayısını yüksek tutarak istediğimiz basamak hassasiyetinde hesap yapabiliriz. Bu problemi sonsuz seri çarpımı olarak, yani,

$$\prod_{n=1}^{\infty}\left(1-\frac{4x^2}{(2n-1)^2\pi^2}\right)$$

şeklinde 5 basamak doğrulukla hesaplamak istersek, programda nasıl bir değişiklik yapmamız gerekir? *Yakınsamanın beş basamak doğrulukla gerçekleştiğini nasıl anlarız?*

11.2 TÜREV FORMÜLLERİN TÜRETİLMESİ

Çok özel durumlarda bir fonksiyonun n.ci dereceden türevini analitik olarak hesaplamak mümkündür. Örneğin,

$$f(x)=\ln x \Rightarrow f^{(n)}(x)=\frac{(-1)^{n-1}(n-1)!}{x^n}$$

$$f(x)=e^{ax} \Rightarrow f^{(n)}(x)=a^n e^{ax}$$

$$f(x)=\sin x \Rightarrow f^{(n)}(x)=\sin(x+\frac{n\pi}{2})\ \ldots$$

Bu şekilde genel türevleri hesaplanabilen fonksiyonların, analitik türevlerini hesaplamak için uygun bir alt program yazılabilir. Örneğin, $f(x)=\sin x$ için

```
FUNCTION turev_sinx(n,x)
IMPLICIT NONE
REAL :: turev_sinx, pi=3.14159
REAL, INTENT(IN)   :: x   ! Arzulanan x için türevi
INTEGER, INTENT(IN):: n   ! Türevin derecesi
turev_sinx=SIN(x+0.5*pi*REAL(n))
END FUNCTION turev_sinx
```

ÖRNEK 3: Aşağıda verilen fonksiyonu, dışarıdan girilen herhangi bir x gerçek sayısı için fonksiyonun kendisini, birinci ve ikinci dereceden türevlerini hesaplayan bir `FUNCTION` programı yazınız.

$$f(x)=\frac{\ln(1+x^3)}{x}$$

Fonksiyonun birinci ve ikinci dereceden türevleri sırasıyla

$$f'(x)=\frac{9x}{1+3x^3}-\frac{\ln(1+x^3)}{x^2}$$

$$f''(x)=\frac{-18}{1+3x^3}+\frac{18-27x^3}{(1+3x^3)^2}+\frac{2\ln(1+x^3)}{x^3}$$

olmaktadır.

Buna göre,

```
PROGRAM Ornek3
IMPLICIT NONE
REAL :: x, Analitik_Turev
INTEGER:: n
READ*, x      ! x değerini gir
DO n=0,2
  PRINT*, n,x,Analitik_Turev(n,x)
END DO
END PROGRAM Ornek3

FUNCTION Analitik_Turev(derece,x)
IMPLICIT NONE
INTEGER, INTENT(IN) :: derece
REAL, INTENT(IN)    :: x
REAL :: Analitik_Turev
SELECT CASE (derece)
    CASE (:-1)  ! Hata yolu
      PRINT*, derece,'.ci dereceden türev tanımsız'
    CASE (0)    ! f(x) fonksiyonunun hesabı
      Analitik_Turev= LOG(1.+3.*x**3)/x
    CASE (1)    ! f'(x) türevinin hesabı
      Analitik_Turev= 9.*x/(1.+3.*x**3)&
           -LOG(1.+3.*x**3)/x**2
    CASE (2)    ! f''(x) türevinin hesabı
      Analitik_Turev=-18./(1.+3.*x**3)+&
     (18.-27.*x**3)/(1.+3.*x**3)**2+2.*&
      LOG(1.+3.*x**3)/x**3
    CASE (3:)   ! n>=3 için türev ?
      PRINT*, derece,'.ci dereceden türev'&
      ' programlanmamıştır... '
END SELECT
END FUNCTION Analitik_Turev
```

Ancak, maalesef, her fonksiyon için n .ci dereceden bu tür analitik türev bağıntıları elde etmemiz mümkün olmamaktadır. Çok uzun ve karmaşık analitik ifadelerin türevleri de çok uzun olabilir veya fonksiyon kesikli dağılım ile belirli olabilir. Bu nedenle analitik türevi hesaplamak için türevin tanımından yararlanmamızı gerekli kılar.

Matematik derslerinden hatırlayacağınız gibi, bir $f(x)$ fonksiyonunun $x=a$ değerindeki türevi tanım olarak

$$f'(a)=\lim_{h\to 0}\frac{f(a+h)-f(a)}{h} \tag{11.1}$$

olarak verilmektedir.

Bilgisayar aracılığıyla türev hesabında, analitik olarak verilen bir $f(x)$ fonksiyonunun türevine ilaveten, analitik ifadesi bilinmeyen sadece kesikli dağılımı bilinen fonksiyonların türevleri de hesaplanabilmektedir.

Örneğin, aşağıda verilen fonksiyon, analitik olarak tanımı bilinen bir fonksiyondur.

$$f(x) = \sin\left(e^x - 2x + x^2\right) + 3x - \arctan(x) \tag{11.2}$$

Diğer taraftan, aşağıdaki tablo ile verilen, suyun buharlaşma gizli ısısının sıcaklıkla değişimi kesikli dağılım vermektedir; bu değişim bir $f(T)$ şeklinde ifade edilememektedir.

T (K)	280	300	320	340	360	380
h_{fg} (kJ/kg)	2485	2438	2390	2341	2290	2248

Şimdi türevin tanımına dönecek olursak, tanımdan çıkartılan sonuç *h* değerini mümkün olduğu kadar küçük seçmek olacaktır. Bu durumda herhangi bir a noktasında $f(x)$ fonksiyonunun türevi

$$f'(a) \cong \frac{f(a+h) - f(a)}{h} \tag{11.3}$$

olarak yazılabilir. Bu türev formülüne *ileri fark formülü* denir ve bu formül ile yapılan hesabın hata oranı h ile doğru orantılıdır. İleri fark formülü olarak adlandırılmasının nedeni de dikkat edileceği gibi a noktasından h kadar ilerideki bir noktaya göre eğim hesaplanmaktadır.

Eğer *a* noktasının *h* kadar gerisindeki bir noktaya göre eğim (yani türev) hesaplanacak olursa, formül

$$f'(a) \cong \frac{f(a) - f(a-h)}{h} \tag{11.4}$$

şeklini alır ve bu formül de *geri fark formülü* olarak adlandırılır. Geri fark formülü ile yapılan hata oranı da aynı şekilde h ile doğru orantılıdır.

Bir başka hesap şekli daha ortaya konabilir. Bu da *a* noktasının *h* kadar ilerisinde ve gerisindeki değerleri esas alan eğim hesabıdır: buna göre

$$f'(a) \cong \frac{f(a+h) - f(a-h)}{2h} \tag{11.5}$$

elde edilir. Bu durumda a noktası $a+h$ ile $a-h$ arasında yer aldığından dolayı, bu formül *merkezi fark formülü* olarak adlandırılır. Merkezi fark formüllerinde yapılan hata ise, h^2 ile doğru orantılıdır.

İkinci dereceden türevlere gelince, $f''(a)$ hesaplamak için ileri-fark formülü olarak

$$f''(a) = \frac{d}{dx}\frac{df(a)}{dx} = \frac{f'(a+h) - f'(a)}{h} = \frac{1}{h}\left(\frac{f(a+2h) - f(a+h)}{h} - \frac{f(a+h) - f(a)}{h}\right)$$
$$= \frac{f(a+2h) - 2f(a+h) + f(a)}{h^2} \tag{11.6}$$

elde edilir. Benzer Şekilde geri ve merkezi fark formülleri sırasıyla

$$f''(a)=\frac{d}{dx}\frac{df(a)}{dx}=\frac{f'(a)-f'(a-h)}{h}=\frac{1}{h}\left(\frac{f(a)-f(a-h)}{h}-\frac{f(a-h)-f(a-2h)}{h}\right)$$
$$=\frac{f(a)-2f(a-h)+f(a-2h)}{h^2} \qquad (11.7)$$

$$f''(a)=\frac{f'(a+h/2)-f'(a-h/2)}{h}=\frac{1}{h}\left(\frac{f(a+h)-f(a)}{h}-\frac{f(a)-f(a-h)}{h}\right)$$
$$=\frac{f(a+h)-2f(a)+f(a-h)}{h^2} \qquad (11.8)$$

elde edilir.

ÖRNEK 4: $f(x)=x^2e^{-x}$ fonksiyonunun $x=1$ 'deki birinci ve ikinci dereceden türevlerini türev tanımından yararlanarak ve h değerini değiştirerek hesaplayan bir program yazınız ve analitik değerleri ile kıyaslayınız.

Söz konusu fonksiyonun türevleri $f'(x)=(2x-x^2)e^{-x}$ ve $f''(x)=(x^2-4x+2)e^{-x}$ analitik ifadelerinde $x=1$ alındığında $f'(1)=0.367879$ ve $f''(1)=-0.367879$ olarak hesaplanır. $x=1$ noktası için ileri ve merkezi fark formülleri, sırasıyla

$$f'(1)=\frac{f(1+h)-f(1)}{h} \quad \text{ve} \quad f'(1)=\frac{f(1+h)-f(1-h)}{2h}$$
$$f''(1)=\frac{f(1+2h)-2f(1+h)+f(1)}{h^2} \quad \text{ve} \quad f''(1)=\frac{f(1+h)-2f(1)+f(1-h)}{h^2}$$

şeklinde yazılır. Türevleri, h =0.1, 0.01 ve 0.001 için, bu formülleri kullanarak hesaplayacak olursak, sonuçların Tablo 11. l 'deki şekilde olduğu görülecektir.

Tablo 11.1 Türev formüllerinin *h* ile değişimi

h	$f'(1)$ (ileri)	$f'(1)$ (merkezi)	$f''(1)$ (ileri)	$f''(1)$ (merkezi)
0.1	0.348945	0.367263	-0.394891	-0.366344
0.01	0.366034	0.367873	-0.371452	-0.367864
0.001	0.367695	0,367879	-0.368246	-0.367879

Tablo 11. l 'deki türev değerleri incelendiğinde, merkezi fark formülleriyle elde edilen türev değerlerinin gerçek değerlere, ileri fark formülleri ile elde edilenlerden, daha yakın olduğu gözlenir. Bunun nedeni kullanılan formüllerin hata miktarıdır; ileri fark formülündeki hata, h=0.l için, $0.1K_1$ (K_1 ve K_2 sabitlerdir) iken merkezi fark formülünde $0.01K_2$ dir. Bütün formüllerde h değeri küçüldükçe K_1 ve K_2 'nin hata terimindeki etkisi küçülmektedir. Bu etki merkezi fark formülünde, K_2 'nin katsayısı daha küçük olduğundan dolayı, daha hızlı olmaktadır.

Bu şekilde analitik olarak verilen bir fonksiyonun herhangi bir $x=a$ noktasında birinci ve ikinci dereceden türevlerini veren bir program kesiti aşağıda verilmektedir.

```
PROGRAM turev
IMPLICIT NONE
!--------------------------------------------------
! ÖRNEĞİN F(X)=X*X*EXP(-X) 'İN TÜREVLERİNİN HESABI
! BU PROGRAMDA
!    * a  = TÜREVİN HESAPLANMASI ARZULANAN x=a
!    * F1 = BİRİNCİ DERECEDEN TÜREV,
!    * F2 = İKİNCİ DERECEDEN TÜREVİ VERMESİ İÇİN
! ATANAN DEĞİŞKENLERDİR. KULLANILAN FORMÜLLER
! MERKEZİ-FARK FORMÜLLERİDİR.
!--------------------------------------------------
REAL :: a=1.0, h, f1, f2
PRIN*, 'h Değerini Giriniz'
READ*, h
f1=0.5*(f(a+h)-f(a-h))/h
f2=(f(a+h)-2.0*f(a)+f(a-h))/h**2
PRINT*, 'Fonksiyonun x=',a,' daki Türevleri '
PRINT*, 'Birinci Dereceden Türev=',f1
PRINT*, 'ikinci  Dereceden Türev=',f2
END PROGRAM turev
!
REAL FUNCTION F(X)
IMPLICIT NONE
REAL :: F
REAL, INTENT(IN) :: x
  F= x*x*EXP(-x)
END FUNCTION f
```

Diğer taraftan mühendislik hesaplarında kullanılan birçok fiziksel değişkenlerin (yoğunluk, gizli ısı, viskozite, vs) sıcaklıkla veya basınçla değişimleri $f(T)$ gibi bir fonksiyon olarak ifade edilmesi mümkün değildir. Bu fonksiyon değerleri farklı sıcaklık ve/veya basınçta deneysel olarak tespit edilmiştir. Bu nedenle her sıcaklık ve/veya basınçtaki özelliği bilemeyiz. Bu tür verilere *kesikli dağılım* adı verilir. Analitik olmayan (kesikli dağılan) fonksiyonların çeşitli $x_0, x_1, \ldots, x_n$ değerlerine karşılık gelen $f_0, f_1, \ldots, f_n$ değerleri olsun. Bu durumda herhangi bir x_i noktası için türev formülleri

$$f_i' = f'(x_i) \cong \frac{f_{i+1} - f_i}{h} \qquad \text{(İleri fark formülü)}$$

$$f_i' = f'(x_i) \cong \frac{f_{i+1} - f_{i-1}}{2h} \qquad \text{(merkezi fark formülü)}$$

şeklinde yazılabilir. Ancak bu formüller belirli bir oranda hata içermektedir. Türevleri başka yollarla hesaplama olanağımız olmadığından belirli oranda hataya razı olacağız ya da mevcutsa daha az hata ile hesaplama tekniklerini, formüllerini kullanacağız.

Dikkat edilecek nokta x_i'lerin eşit aralıklarla dağılmış olmalarıdır; yani $x_i = x_{i-1} + h$ şeklinde belirtilebilmesidir. Bu tür dağılımların türevlerini hesaplayan bir program yazarken f_i'lerin bir boyutlu indisli değişken olarak tanımlanmaları büyük kolaylık sağlar. Aşağıda herhangi bir x_i noktasında birinci ve ikinci dereceden türevi hesaplayan bir `SUBROUTINE` verilmektedir.

```
SUBROUTINE turev(n, m, x, f, f1, f2)
IMPLICIT NONE
!---------------------------------------------------
!  EŞİT ARALIKLARLA KESİKLİ DAĞILIMI OLAN DEĞİŞKENLERİN
!  TÜREV HESABI
!---------------------------------------------------
INTEGER, INTENT(IN)  :: n  ! Kesikli dağılım uzunluğu
INTEGER, INTENT(IN)  :: m  ! Türevi alınacak noktanın indisi
REAL, DIMENSION(n), INTENT(IN) :: x  ! Fonksiyonun absisleri
REAL, DIMENSION(n), INTENT(IN) :: f  ! Fonksiyon değerleri
REAL, INTENT(OUT)  :: f1   ! Birinci dereceden türev
REAL, INTENT(OUT)  :: f2   ! İkinci dereceden türev
REAL :: h, h2
INTEGER :: i
h  = x(2)-x(1)              ! Aralık genişliğinin hesabı
h2 = h*h
! Merkezı-Fark Formülleri Kullanılmıştır.
f1 =0.5*(f(m+1)-f(m-1))/h
f2 =(f(m+1)-2.0*f(m)+f(m-1))/h2
END SUBROUTINE turev
```

Çeşitli uygulamalara referans teşkil etmesi bakımından, bazı türevler için fark formülleri aşağıdaki verilmiştir.

Merkezi fark formülleri (hata h^2 ile orantılı):

$$f_i' = \frac{f_{i+1} - f_{i-1}}{2h}$$

$$f_i'' = \frac{f_{i+1} - 2 f_i + f_{i-1}}{h^2}$$

$$f_i''' = \frac{f_{i+2} - 2 f_{i+1} + 2 f_{i-1} - f_{i-2}}{2h^3}$$

İleri Fark Formülleri:

$$f_i' = \frac{f_{i+1} - f_i}{h}, \quad h \text{ ile orantılı}$$

$$f_i' = \frac{-f_{i+2} + 4 f_{i+1} - 3 f_i}{2h}, \quad h^2 \text{ ile orantılı}$$

Geri Fark Formülleri:

$$f_i' = \frac{f_i - f_{i-1}}{h}, \quad h \text{ ile orantılı}$$

$$f_i' = \frac{3 f_i - 4 f_{i-1} + f_{i-2}}{2h}, \quad h^2 \text{ ile orantılı}$$

ALIŞTIRMALAR

11.1 Birden girilen herhangi bir `n` tamsayısına kadar (a) tüm sayıların (b) sadece tek olanların (c) sadece çift olanların toplamını veren bir program yazınız.

11.2 Birden girilen herhangi bir `n` tamsayısına kadar (a) tüm sayıların (b) sadece tek olanların (c) sadece çift olanların küpleri toplamını veren bir program yazınız.

11.3 Programa girilen herhangi bir `[a,b]`aralığında 0.1 artırımlarla elde sayı dizisinin (a) toplamını (b) kareleri toplamını (c) küpleri toplamını bir program yazınız.

11.4 Aşağıda verilen serinin toplamını 4 basamak doğrulukla bulmak için bir program yazınız.

$$\frac{1}{2^2+1}+\frac{2}{2^3+1}+\frac{3}{2^4+1}+\frac{4}{2^5+1}+\cdots$$

11.5 Aşağıdaki seri ifadesi ile verilen `T` değerini, programa dışarıdan girilecek herhangi bir x ($0<x<2$) değeri için hesaplamak istiyoruz. Bu ifadeyi hesaplayan bir programı yazınız. *NOT:* Programınız, girilen x değerinin belirtilen aralıkta olup olmadığını kontrol eden ve kullanıcıyı uyaran bir kısım içersin.

$$T=\sum_{i=1}^{5}\sum_{j=0}^{8} a_{i,j}x^i\left(1-x\right)^j \quad \text{ve} \quad a_{i,j}=\frac{i^2+j^2}{\sqrt{i^4+j^4}}$$

11.6 Aşağıdaki seri işlemini verilen şartlar altında hesaplamak için bir akış şeması çiziniz ve program yazınız.

$$\sum_{i=1}^{17} a_i b_i = ? \qquad a_m=\begin{cases} m\sin(\pi m/8), & m\le 8 \\ m^2\sin(\pi m/32), & m>8 \end{cases} \qquad b_n=\begin{cases} e^{n/10}, & n<6 \\ n^2\sin(n\pi/5), & n\ge 6 \end{cases}$$

11.7 Aşağıdaki işlemi herhangi bir x değeri için hesaplayan programın akış şeması çiziniz ve programını yazınız. *NOT:* Burada kullanılan d_i'ler dışarıdan temin edilen sabitlerdir.

$$\prod_{i=1}^{9} d_i x \cos ix$$

11.8 Tanjant hiperbolik fonksiyonun değerini hesaplamak için verilen bir algoritma aşağıda çıkartılmıştır.

$x<0$ için $\tanh x=-\tanh x,$

$|x|\le 0.00034$ için $\tanh x = x,$

$0.00034<|x|<0.17$ için $\tanh x = f\left[A+f^2\left(B+C\left(D+f^2\right)^{-1}\right)\right]^{-1}$

$0.17<|x|<11.15$ için $\tanh x=\left(e^x-e^{-x}\right)/\left(e^x+e^{-x}\right)$

$|x|>11.15$ için $\tanh x = 1$

Burada $f=4x\log_2 e$, A=5.7707802, B=0.0173286, C=14.1338411 ve D=349.6699888 olarak verilen sabitlerdir. Verilen algoritma uyarınca tanh(x) fonksiyonunu hesaplayan bir

`FUNCTION Tanh2` isimli bir alt program yazınız; arşiv fonksiyonu `TANH(x)` ile test ediniz. *NOT:* Dışarıdan girilen bir gerçek `x` sayısı için `Tanh2(x)` ve `Tanh(x)` ile mutlak hata `Tanh(x)-Tanh2(x)` hesaplanacaktır.

11.9 Aşağıdaki matematiksel işlemleri yapmak üzere hazırlayacağınız bir programa önce bir algoritma hazırlayın, bu algoritma için akış şeması çiziniz ve program yazınız.

$$\sum_{i=1}^{10} y_i \left(\sum_{j=1}^{15} c_j x_i^j \right)$$

Burada $y_i = 0.025(i^3 + i^2)$, $c_j = \left[3^j + (-4)^j \right] \big/ 8^j$ ve $x_i = i/15$ olarak verilmektedir.

11.10 Aşağıdaki sonsuz seri çarpımı ifadelerini dışarıdan temin edilecek herhangi bir *x* değeri için hesaplayan bir programlar yazınız.

(a) $xe^x \left\{ \left(1+\frac{x}{1}\right)e^{-x} \right\} \left\{ \left(1+\frac{x}{2}\right)e^{-x/2} \right\} \left\{ \left(1+\frac{x}{3}\right)e^{-x/3} \right\} \left\{ \left(1+\frac{x}{4}\right)e^{-x/4} \right\} \cdots$

(b) $x\left(1+\frac{x^2}{2!2^2}\right)\left(1+\frac{x^4}{4!2^4}\right)\left(1+\frac{x^6}{6!2^6}\right)\left(1+\frac{x^8}{8!2^8}\right)\cdots$

(c) $x\left(1+\frac{1}{x^2}\right)\left(1+\frac{1\cdot 3}{x^4}\right)\left(1+\frac{1\cdot 3\cdot 5}{x^6}\right)\left(1+\frac{1\cdot 3\cdot 5\cdot 7}{x^8}\right)\cdots$

11.11 Bir şahıs yıllık `i` faiz veren bir bankaya her yılın aynı gününde `A` YTL'nı `n` yıl boyunca yatırmaktadır. `n` yılın sonunda biriken para `F` faizi ile beraber

$$F = A\left[\left(1+\frac{i}{100}\right) + \left(1+\frac{i}{100}\right)^2 + \left(1+\frac{i}{100}\right)^3 + \cdots + \left(1+\frac{i}{100}\right)^n \right]$$

formülü ile hesaplanıyor. *A*, *i* ve *n* değişken olmak üzere faizi hesaplayan bir program yazın. *NOT:* Yukarıdaki formülün genel şekli:

$$F = A\left[\left(1+\frac{i}{100m}\right)^m + \left(1+\frac{i}{100m}\right)^{2m} + \cdots + \left(1+\frac{i}{100m}\right)^{nm} \right]$$

Burada *m* bir yıldaki faiz periyodu sayısıdır.

11.12 Aşağıda seri şeklinde verilen ifade *x*'in [1,3] kapalı aralığında 0.1'lik artırımlarla, dört basamak doğrulukla (hata mertebesi $\varepsilon = 10^{-4}$'ten küçük olacak şekilde) hesaplanmak isteniyor. *NOT:* Seri toplamının bir `FUNCTION` alt programıyla yapılması önerilmektedir.

$$x + \left(\frac{x+2}{8\cdot 1^2+1}\right) + \left(\frac{x+4}{8\cdot 2^2+1}\right) + \cdots + \left(\frac{x+2m}{8\cdot m^2+1}\right) + \cdots = \sum_{n=0}^{\infty} \frac{x+2n}{8\cdot n^2+1}$$

11.13 Dizinin değerleri programa dışarıdan temin edilen (konsoldan veya bir kütükten de olabilir) *N* tane x_i ve *M* tane y_j indisli değişkenlerinden hareket ederek, bu iki dizinin varyanslarını ve toplam varyansını hesaplamak için bir program yazınız. Verilenler:

$$VAR(X) = SS^2 = \sum_{i=1}^{N} \left(x_i - \bar{x}\right)^2 \Big/ N \qquad \text{Toplam varyans} = \frac{N \times VAR(X) + M \times VAR(Y)}{N+M}$$

11.14 Bir lokantanın vestiyerine şapkasını teslim eden K kişi şapkalarını geri almak için vestiyere gittiklerinde vestiyer çalışanı herkesin şapkasını karıştırmış olduğunu fark ediyor. Bu K şahısın şapkalarını almama olasılığı aşağıdaki formül ile saptandığı bilindiğine göre bu problemi, programa temin edilecek bir K değeri için çözecek bir program yazınız.

$$p = \frac{1}{2!} - \frac{1}{3!} + \frac{1}{4!} - \frac{1}{5!} + \cdots \pm \frac{1}{K!} = \sum_{n=2}^{K} (-1)^n \frac{1}{n!}$$

11.15 Herhangi bir x değeri ve $N = 10$ için $Z(x)$ fonksiyonunun değerini aşağıda verilen seri açılımları, n ve x'in fonksiyonu olarak verilen $F_n(x)$ ve $Y_n(x)$ yardımıyla, hesaplamak istiyoruz. Bu işlemi bilgisayar aracılığıyla yapabilmemiz için bir program yazınız.

$$F_n(x) = 1 + \sum_{m=1}^{n} (-1)^m \frac{1 \cdot 3 \cdot 5 \cdots (2m-1)}{(\pi x^2)^{2m}} \qquad Y_n(x) = \log_{10}(x) - \sum_{m=1}^{n} \frac{2 \cdot 4 \cdot 6 \cdots (2m)}{(\pi x^2)^{m+1}}$$

$$Z_n(x) = \sum_{n=1}^{N} \frac{F_n(x) Y_n(x)}{(n+1)!}$$

NOT: Hazırlayacağınız program $F_n(x)$, $Y_n(x)$ fonksiyonlarını ve faktoriyel hesabını yapan üç adet `FUNCTION` içermelidir.

11.16 Aşağıdaki seri işlemlerini ve Z yi hesaplayan bir programı yazınız.

$$x = \sum_{k=1}^{17} \frac{k^2 + 1}{4k^3 + k} \qquad y = \sum_{n=1}^{10} \frac{n^n}{n!} \qquad Z = \frac{xy}{x + y}$$

11.17 Dikdörtgensel basit destekli plakaya uygulanan üniform yük ile plakanın, $0 \le x \le a$ ve $-b/2 \le y \le b/2$, herhangi bir (x, y) noktasındaki bükülme momentleri M_x, M_y ve M_{xy} aşağıdaki seriler ile veriliyor.

$$M_x = Q\pi^2 a^2 \sum_{n=1,3}^{\infty} \left\{ \begin{bmatrix} \dfrac{4}{n^3 \pi^5} + n^2 A_n (1-\nu) \cosh \beta_n y \\ + n^2 B_n \left((1-\nu) \beta_n y \sinh \beta_n y - 2\nu \cosh \beta_n y \right) \end{bmatrix} \sinh \beta_n x \right\}$$

$$M_y = Q\pi^2 a^2 \sum_{n=1,3}^{\infty} \left\{ \begin{bmatrix} \dfrac{4\nu}{n^3 \pi^5} - n^2 A_n (1-\nu) \cosh \beta_n y \\ - n^2 B_n \left((1-\nu) \beta_n y \sinh \beta_n y + 2\nu \cosh \beta_n y \right) \end{bmatrix} \sinh \beta_n x \right\}$$

$$M_{xy} = (1-\nu) Q\pi^2 a^2 \sum_{n=1,3}^{\infty} \left[n^2 A_n \sinh \beta_n y + n^2 B_n \left(\sinh \beta_n y + \beta_n y \cosh \beta_n y \right) \right] \cosh \beta_n x$$

ayrıca

$$\beta_n = \frac{n\pi}{a}, \quad \alpha_n = \frac{\beta_n g}{2}, \quad B_n = \frac{2}{\pi^5 m^5 \cosh \alpha_n}, \quad A_n = -B_n \left(\alpha_n \tanh \alpha_n + 2 \right)$$

olarak tanımlıdır. Burada Q uygulanan üniform yük ve ν poisson oranıdır. Plaka üzerindeki herhangi bir noktadaki momentleri hesaplayan bir program yazınız. *Veriler:* Q=25 kg, ν=0.3, $a = 20cm$, $b = 24cm$ ve $\varepsilon = 0.0001$.

11.18 Aşağıdaki formül ile verilen α ifadesini hesaplayan bir program yazınız.

$$C_k = \cosh\left(k^{1/2} - k^{1/3}\right) \qquad D_k = \frac{\arccos\left(k/50\right)}{\sqrt{3k^2 - 1}}$$

$$\alpha = \sqrt{1 + \sum_{n=1}^{17} C_n D_n} + \sum_{k=1}^{9}\left(k + \prod_{n=0}^{k}\frac{1}{(2n+1)}\right)^{-1}$$

11.19 Bir araba harekete geçtikten sonra her 2.5 sn'de bir hızı ölçülerek kaydedilmiş ve bu ölçümler aşağıdaki tabloda verilmiştir. Bu arabanın 25 sn sonu itibarıyla her 2.5 sn'ye denk gelen ivmesini hesaplayan bir program yazınız (*NOT:* İvme $a = dv/dt$ ile hesaplanacaktır).

t (sn)	0	2.5	5	7.5	10.0	12.5	15	17.5	20	22.5	25
v (m/sn)	0	13	35	47	45	53	46	48	36	25	0

11.20 Aşağıdaki verilerde her bir x noktası için birinci ve ikinci dereceden türevlerini hata oranı en az olacak şekilde hesaplayınız.

x_i	0	0.5	1.0	1.5	2.0	2.5
y_i	1	0.8	0.2	0.25	0.31	0.44

11.21 $f\left(x\right) = 3\sqrt{x+1-x^2}\Big/\ln\left(x+1\right)$ fonksiyonunun türevim $\left(f'(1.2)\right)$ $x = 1.2$ 'de hata oranı $\left(0.1\right)^2$ ile orantılı olacak şekilde ileri, geri ve merkezi fark formülleri ile hesaplayınız ve analitik olarak hesaplanan türev değeri (yani, gerçek değeri) ile karşılaştırınız. Yazacağınız bir programda *h* girdi olsun.

11.22 Aşağıda dağılımı verilen fonksiyonun f_i' ve f_i'' türevlerini mümkün olan en az hata ile hesaplayınız.

x	1.2	1.5	1.9	2.3	2.4	2.8
$f(x)$	1	1.8	2.2	2.25	3.1	3.44

11.23 *f*(*x*) fonksiyonu aşağıdaki şekilde tanımlanmıştır:

$$f\left(x\right) = \log_3\left(\frac{\sin x^3}{x^3 - \arctan x^3}\right) + x^3$$

[1,2] aralığında her 0.1'lik artırımlara karşılık gelen *x* değerleri için fonksiyonun birinci ve ikinci dereceden türevlerini merkezi fark formülleri ile hesaplayan bir program yazınız. Programda $h = 0.001$ alınız ve herhangi bir *x* için türevleri hesaplayan bir `SUBROUTINE` oluşturunuz.

11.24 *n* ci dereceden bir polinomun, dışarıdan girilen herhangi bir *x* gerçek sayısı için, y', y'' ve y''' türevlerini analitik olarak hesaplayan bir `FUNCTION Fonk(n,derece,x,a)`

programı yazınız. Burada `n` polinomun derecesi, `derece` kaçıncı dereceden türevin hesaplanacağı, `a` polinomun katsayılarıdır (`a(0:n)`). Fonksiyonda `Derece=0` ise polinomun kendisini, `Derece=1` ise birinci dereceden türevini, vs getirecektir.

$$\text{Fonk}(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = \sum_{i=0}^{n} a_i x^i$$

İPUCU: Fonksiyonu sonlu toplam şeklinde yazdıktan sonra,

$$\text{Fonk}'(x) = a_1 + 2a_2 x + 3a_3 x^2 + \cdots + n a_n x^{n-1} = \sum_{i=1}^{n} i a_i x^{i-1}$$

olduğuna dikkat ediniz. Aynı işlemi ikinci ve üçüncü dereceden türevler için benzer şekilde tekrar ediniz.

11.25 Aşağıda verilen fonksiyonun, dışarıdan girilen herhangi bir x gerçek sayısı için, y', y'' ve y''' türevlerini analitik olarak hesaplayan bir `FUNCTION Fonk(x,derece)` programı yazınız. Burada `derece` kaçıncı dereceden türevin hesaplanacağı, `derece=0` ise fonksiyonun kendisini, `Derece=1` ise birinci dereceden türevini, vs getirecek bir program yazınız. *NOT:* Fonksiyonun söz konusu türevleri analitik olarak elle hesaplanıp, programlanacaktır.

$$f(x) = x^3 \sin^{12} x$$

BÖLÜM 12
KÖK BULMA

Mühendis ve bilim adamlarının belki de en çok ihtiyaç duydukları matematiksel işlemlerden biri de bir fonksiyonun kökünü (veya köklerini) bulmaktır. Kök bulma işleminde fonksiyonlar karşımıza, polinom şeklinde ya da trigonometrik, üstel, hiperbolik v.b fonksiyonların kombinasyonunu içeren şekilde çıkabilmektedir. Bunlardan

$$x \tan x - x = \frac{1}{2} \ , \quad e^x \sin x = 1 \ , \quad x^2 e^x - 2x = \tan x$$

gibi ifadelerin köklerini analitik yöntemler ile bulmak ya çok zor yada imkansızdır. Bu tür fonksiyonlara *lineer olmayan denklemler* adı verilir. Bu denklemleri sayısal yöntemler kullanarak, özellikle bilgisayar yardımıyla çözmek gerekir.

Polinom şeklinde olan fonksiyonlardan en basiti ve herkes tarafından bilinen ikinci dereceden denklemin köklerinin hesabını içerir, ikinci dereceden denklem

$$a_2 x^2 + a_1 x + a_0 = 0$$

genel halinde verilmişse, bu denklemin kökleri $\Delta = a_1^2 - 4a_2 a_0$ olan diskriminantın işaretine bağlı olarak gerçek yada sanal/komplekstir.

Eğer $\Delta > 0$ ise iki gerçek kök mevcuttur.

$$x_{1,2} = \frac{-a_1 \pm \sqrt{\Delta}}{2a_2}$$

Eğer $\Delta = 0$ ise bir birine eşit iki gerçek kök mevcuttur.

$$x_{1,2} = \frac{-a_1}{2a_2}$$

Eğer $\Delta < 0$ ise bir birinin eşleniği şeklinde iki sanal kök mevcuttur.

$$x_{1,2} = \frac{-a_1 \pm i\sqrt{-\Delta}}{2a_2}$$

ikinci dereceden bir denklemin köklerini veren, çift hassasiyetle işlem yapan, `ikinci` isimli bir alt program ile bunun kullanıldığı ana program aşağıda verilmiştir:

```
PROGRAM ikinciDDenklem
IMPLICIT NONE
REAL(KIND=8), DIMENSION(2):: x1, x2
REAL(KIND=8) :: a, b, c
PRINT*, "İkinci dereceden denklemin katsayılarını"
PRINT*, "a, b ve c sırasıyla giriniz"
READ*, a,b,c
CALL ikinci(a,b,c,x1,x2)
! Burada Kökler Ekrana Yazdırılıyor
```

```
PRINT*, " KÖKLER "
WRITE(*,10) x1(1),x1(2)
WRITE(*,20) x2(1),x2(2)
10 FORMAT(3X,'X1=(',f12.6,2X,',',f12.6,' i)')
20 FORMAT(3X,'X2=(',f12.6,2X,',',f12.6,' i)')
END PROGRAM ikinciDDenklem

SUBROUTINE ikinci(a2,a1,a0,x1,x2)
IMPLICIT NONE
REAL(KIND=8), INTENT(IN) :: a0, a1, a2
REAL(KIND=8), DIMENSION(2), INTENT(OUT):: x1(2), x2(2)
REAL(KIND=8) :: delta
!
!  İKİNCİ DERECEDEN DENKLEMİN KÖKLERİNİ HESAPLAYAN
!  ALT PROGRAM.  İKİNCİ DERECEDEN DENKLEM
!          2
!   A2 * X + A1 * X + A0 = 0
!
!  ŞEKLİNDEDİR.
!
!  KÖKLER
!        Kök1 = x1(1) + x1(2) i
!        Kök2 = x2(1) + x2(2) i
!  ŞEKLİNDE OLACAKTIR. NOT: BU PROGRAMIN 'COMPLEX'
!  TANIMI ILE PROGRAMLANMASI DAHA UYGUNDUR.
!  ====  Not: Çift hassasiyetli işlem yapar!!!!  ====
delta=a1*a1-4.0*a0*a2
x1(2)=0.d0  ! Sanal kısımları sıfırla
x2(2)=0.d0
IF(delta>=0.d0) THEN
     delta=DSQRT(delta)    ! Gerçek kısımları hesapla
     x1(1)=0.5d0*(-a1+delta)/a2
     x2(1)=0.5d0*(-a1-delta)/a2
ELSE
     delta=DSQRT(-delta)  ! Gerçek kısımları hesapla
     x1(1)=-0.5d0*a1/a2
     x2(1)=x1(1)
     x1(2)=0.5d0*delta/a2 ! Sanal kısımları hesapla
     x2(2)=-x1(2)
END IF
END SUBROUTINE ikinci
```

Diğer taraftan, üçüncü dereceden bir polinomun köklerinin bulunması için de bir algoritma geliştirilmiştir. Bu algoritma, üçüncü dereceden polinom

$$a_3x^3 + a_2x^2 + a_1x + a_0 = 0$$

şeklinde veriliyorsa Q, R, S ve T değerleri

$$Q = \frac{1}{3}\left(\frac{a_1}{a_3} - \frac{a_2^2}{3a_3^2}\right), \qquad R = \frac{1}{2}\left(\frac{a_2 a_1}{3a_3^2} - \frac{a_0}{a_3} - \frac{2a_2^3}{27a_3^3}\right),$$

$$S = \left[R + \sqrt{Q^3 + R^2}\right]^{1/3}, \qquad T = \left[R - \sqrt{Q^3 + R^2}\right]^{1/3}$$

ifadelerinden hesaplanır. Eğer $a_0,\ a_1,\ a_2,\ a_3$ katsayıları gerçek sayı ve $D = Q^3 + R^2$ diskriminant ise, buradan,

(1) $D > 0$ ise, bir kök gerçekl diğer iki kök kompleks eşleniktir;

$$x_1 = S + T - \frac{a_2}{3a_3}$$

$$x_2 = -\frac{1}{2}(S + T) - \frac{a_2}{3a_3} + \frac{1}{2} i\sqrt{3}(S - T)$$

$$x_3 = -\frac{1}{2}(S + T) - \frac{a_2}{3a_3} - \frac{1}{2} i\sqrt{3}(S - T)$$

(2) $D = 0$ ise, bütün kökler gerçektir ancak son iki kök birbirine eşit katlı köktür;

$$x_1 = S + T - \frac{a_2}{3a_3}$$

$$x_{2,3} = -\frac{1}{2}(S + T) - \frac{a_2}{3a_3}$$

(3) $D < 0$ ise, bütün kökler gerçek ancak birbirine eşit değildir:

$$x_1 = 2\sqrt{-Q}\cos\left[\frac{1}{3}\cos^{-1}\left(\frac{R}{\sqrt{-Q^3}}\right)\right] - \frac{a_2}{3a_3}$$

$$x_2 = 2\sqrt{-Q}\cos\left[\frac{1}{3}\cos^{-1}\left(\frac{R}{\sqrt{-Q^3}}\right) + \frac{2\pi}{3}\right] - \frac{a_2}{3a_3}$$

$$x_3 = 2\sqrt{-Q}\cos\left[\frac{1}{3}\cos^{-1}\left(\frac{R}{\sqrt{-Q^3}}\right) + \frac{4\pi}{3}\right] - \frac{a_2}{3a_3}$$

Maalesef bütün polinom/fonksiyon kombinasyonları için yukarıda verilen şekilde güzel analitik çözüm algoritmaları bulmak mümkün değildir. Sadece bazı özel fonksiyonlar için ortaya atılan metotlar vardır.

12.1 POLİNOMLARIN TÜM KÖKLERİNİN HESAPLANMASI

Aşağıdaki şekilde genel olarak verilen bir polinomu ele alalım:

$$a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \cdots + a_1 x + a_0 = 0$$

Bu polinomun köklerinin bulunması için birkaç metottan birisi olan *Bairstow* metodu iteratif bir çözüm yöntemidir. Yöntem polinomu $x^2 + px + q$ ikici dereceden ifadesine bölünmesini sağlayan p ve q katsayılarını bulmak ve daha sonra bu ikinci dereceden denklemin köklerini hesaplamak esasına dayanır. Polinomun derecesi tek ise son kalan birinci dereceden ifadenin kökü gerçektir. Yöntemin sayısal çözüm algoritması aşağıdaki verilmektedir:

1. Polinomun derecesini (n) ve katsayılarını A_i $(i = n, n-1, \ldots, 1, 0)$ gir.
2. n=1 ise kök tektir ve $x = -a_0 / a_1$'e eşittir. Adım 11'e git
3. n=2 ise ikinci dereceden $a_2 x^2 + a_1 x + a_0 = 0$ denklemdir. $p = a_1 / a_2$, $q = a_0 / a_2$ olarak ata ve Adım 11'e git
4. $n \geq 3$ ise $p = 0$ ve $q = 0$ tahmin değerleri olarak al
5. $B_0 = B_1 = 0$ değerleri ile başla ve $i = 2, 3, \cdots, (n+2)$ için
   $B_i = A_{i-2} - pB_{i-1} - qB_{i-2}$ katsayılarını hesapla
6. $C_0 = C_1 = 0$ değerleri ile başla ve $i = 2, 3, \cdots, (n+2)$ için
   $C_i = B_{i-1} - pC_{i-1} - qC_{i-2}$ katsayılarını hesapla
7. Adım 3 ve 4'de hesaplanan katsayıları kullanarak, aşağıdaki artırım ifadeleri hesapla
   $$\Delta p = \frac{C_n B_{n+2} - B_n C_{n+1}}{C_{n+1}^2 - C_n (B_{n+1} + C_{n+2})} \qquad \Delta q = \frac{-B_{n+1}(qC_n + pC_{n+1}) - B_{n+2} C_{n+1}}{C_{n+1}^2 - C_n (B_{n+1} + C_{n+2})}$$
8. p ve q'nun yeni tahmin değerlerini hesapla (k iterasyon adımıdır)
   $$p^{(k+1)} = p^{(k)} + \Delta p \qquad q^{(k+1)} = q^{(k)} + \Delta q$$
9. Yakınsama kriterini kontrol ederek, p ve q'nun yakınsamasını kontrol et.
   $\dfrac{|\Delta p| + |\Delta q|}{\left|p^{(k+1)}\right| + \left|q^{(k+1)}\right|} < \varepsilon$ ise yakınsama sağlanmadı ise Adım 4'e git.
10. Adım 9'daki kritere göre yakınsama sağlandı. Yani n.ci dereceden polinomun çarpanlarından birisi $x^2 + px + q = 0$ ikinci dereceden denklemidir. Bu denklemin köklerini bul.
11. Kök(leri) ekrana veya kütüğe yazdır.
12. Polinomun çarpanlarından birisi $x^2 + px + q = 0$ olduğunda göre $n-2$'nci dereceden çarpım polinomunu bul. $n \leftarrow n-2$ ve $j = 0, 1, 2, ..., n$ için $A_i = B_{i+2}$ ataması ile yeni polinomun katsayılarını bul
13. $n \leq 0$ ise tüm kökler bulunmuştur! Programı sonlandır! $n > 0$ ise Adım 2'ye git.

İkinci dereceden denklemin köklerini bulmak için bu kısımda verilmiş olan `ikinci` isimli alt programından yararlanılmıştır.

```
PROGRAM TestBairstow
IMPLICIT NONE
REAL(KIND=8), DIMENSION(0:20) :: A
INTEGER :: i,n
  WRITE(*,10)
  READ*, n                   ! Polinomun derecesini Oku
    DO i=0,n
       WRITE(*,20) n-i       ! Polinomun katsayılarını oku

       READ*, A(i)
    END DO
  CALL Bairstow(n,A)         ! Çözdürücü programı çağır
10 format(' Polinomun derecesini giriniz: ')
20 format(3x,'A(',i2,')=')
END PROGRAM TestBairstow
```

```
SUBROUTINE Bairstow(k,A)
!-----------------------------------------------------
!  BAIRSTOW Metodu ile k.cı dereceden bir polinomun
!  tüm (gerçek ve sanal) köklerini hesaplayan (ÇİFT
!  HASSASİYETLİ) bir alt programdır.
!
!    k : Polinomun derecesi
!    A : Aşağıdaki düzende verilen polinomun katsayıları
!
!      k       k-1          2
!   A x + A    x    +...+ A x +A x+A ==0
!    k     k-1            2    1   0
!------------------------------------------------------
INTEGER, INTENT(IN) :: k
REAL(KIND=8), DIMENSION(0:k), INTENT(INOUT) :: A
REAL(KIND=8), DIMENSION(0:20) :: B, C, P
REAL(KIND=8), DIMENSION(2) :: x1, x2
REAL(KIND=8) :: deltP,deltQ,d,r,t,u,v,x,y,z,hata
INTEGER::  i, j, m
REAL(KIND=8) :: p1=0.d0, q =0.d0
INTEGER, PARAMETER :: Maxit=100
 m=k     !  M'i hata tahmininde kullanmak için sakla
 n=k
 ! Polinomun katsayıları A(i) ==> P(i) kopyala
 DO i=0, n
    P(i)=A(i)
 END DO
 ! Çözümleme kısmı
 WRITE(*,20)
 WRITE(*,25)
 DO
  SELECT CASE (n)
      CASE (1)       ! n=1 için çözüm
         x = -A(1)/A(0)
         y = 0.d0
         CALL Kok_Yazdir(m,x,y,P)
         EXIT
      CASE (2)       ! n=2 için çözüm
         p1= A(1)/A(0)
         q = A(2)/A(0)
         CALL ikinci(1.d0,p1,q,x1,x2)
         CALL Kok_Yazdir(m,x1(1),x1(2),P)
         CALL Kok_Yazdir(m,x2(1),x2(2),P)
         EXIT
      CASE (3:)      ! n>=3 için çözüm
        j=0
        DO i=1,n
           p(i)=a(i)
        END DO
```

```
          DO
           IF(j>Maxit) THEN
              WRITE(*,30)
              EXIT
           ELSE
              j=j+1
              B(0)=0.d0 ! B(i) ve C(i)'leri hesapla
              B(1)=0.d0
              C(0)=0.d0
              C(1)=0.d0
              DO i=2, n+2
                 B(i)=A(i-2)-p1*B(i-1)-q*B(i-2)
                C(i)=-B(i-1)-p1*C(i-1)-q*C(i-2)
              END DO
              x=B(n+1)  ! deltP ve deltQ'i hesapla
              y=B(n+2)
              z=C(n)
              t=C(n+1)
              u=C(n+2)
              d=t*t-z*(u+x)
              IF (d==0.d0) THEN
                 WRITE(*,30)
                 EXIT
              ELSE
                 deltP=(z*y-x*t)/d
                 deltQ=(-x*(q*z+p1*t)-y*t)/d
                 p1=p1+deltP   ! Yeni p1 ve q
                 q=q+deltQ
                 hata=(dabs(deltP)+dabs(deltQ))&
                      /(dabs(p1)+dabs(q))
              END IF
              IF (hata<EPSILON(1.0)) EXIT
           END IF
          END DO
          call ikinci(1.d0,p1,q,x1,x2)
          call Kok_Yazdir(m,x1(1),x1(2),P)
          call Kok_Yazdir(m,x2(1),x2(2),P)
          n=n-2      ! Polinomun katsayılarını güncelle
          DO i=0, n
             A(i)=B(i+2)
          END DO
        CASE (:0)    ! n<0 ise döngüden çık
          EXIT
    END SELECT
   END DO
   WRITE(*,25)
  20 FORMAT(13x,'Kökler',/,2x, &
     'Gerçek Kısım',4x,'Sanal Kısım',5x,'Mutlak Hata')
  25 FORMAT(2x,44('-'))
  30 FORMAT(' İşlemler Yakınsamıyor...')
  END SUBROUTINE Bairstow
```

```
SUBROUTINE Kok_Yazdir(m,x,y,P)
IMPLICIT NONE
INTEGER, INTENT(IN):: m
REAL(KIND=8), DIMENSION(0:20), INTENT(IN):: P
REAL(KIND=8), INTENT(IN):: x,y
REAL(KIND=8), PARAMETER :: kucuk=1.d-12
REAL(KIND=8) :: a1,r,u,v
INTEGER:: i
! Yapılan hatayı hesapla
  u=P(0)
  v=0.d0
  DO i=1, m
    r=u*x-v*y
    v=u*y+v*x
    u=r+P(i)
  END DO
  a1=DSQRT(u*u+v*v)
  u=0.d0
  v=0.d0
  DO i=1, m
    r=u*x-v*y
    v=u*y+v*x
    u=r+(m-i+1)*P(i-1)
  END DO
  IF (DABS(a1)<kucuk) THEN
    a1=0.d0
  END IF
  IF (u*u+v*v>kucuk) THEN
    a1=a1/DSQRT(u*u+v*v)
  ELSE
    a1=1.d12
  END IF
  WRITE(*,20)  x, y, a1
20 FORMAT(f13.8,3x,f13.8,3x,es14.7)
END SUBROUTINE Kok_Yazdir
```

12.2 LİNEER OLMAYAN DENKLEMLERİN ÇÖZÜMÜ

Çözümü en zor gibi gözüken problemlerden birisi de kökü araştırılan fonksiyonun lineer olmamasıdır; yani yukarıda belirtildiği gibi değişik fonksiyonların kombinasyonu şeklinde olmasıdır: $xe^x - x + 1 = 0$ gibi. Bu durumda, analitik çözüm aramak zamanı boşa harcamak anlamına gelir. Fakat bu tür fonksiyonların da köklerini bulmak için çeşitli sayısal metotlar geliştirilmiştir. Bunlardan en yaygın ve etkin bir şekilde kullanılanı *Newton-Raphson* metodudur.

12.2.1 NEWTON-RAPHSON METODU

Bu metod bir ardışık tekrar gerektirir. Fonksiyonun bir kökü için $x^{(0)}$ tahmin değeri atamamız ve bu tahmin değerini her keresinde yenilememiz gereklidir. Bulunan yeni değerler

fonksiyonda yerine konarak fonksiyonun değerinin sıfıra yaklaşıp yaklaşmadığı kontrol edilir. Eğer fonksiyon sıfıra arzu ettiğimiz bir ε değeri kadar yaklaşırsa çözüm bulunmuş demektir.

Ardışık tekrar için kullanılması tavsiye edilen işlem:

$$x^{(n+1)} = x^{(n)} - \frac{f\left(x^{(n)}\right)}{f'\left(x^{(n)}\right)}$$

ile verilmektedir. Burada ifadede üstel şekilde (n) gösterimin anlamı n.ci işlem aşamasındaki x değerini belirtmektedir.

Newton-Raphson metodunu programlamak için aşağıda bir algoritma verilmektedir:

1. Kök için tahmin değeri, $x^{(0)}$ ve kökün hesaplanması arzulanan basamak hassasiyetini (ε) girdi olarak sapta
2. Fonksiyonun $f\left(x^{(n)}\right)$ ve $f'\left(x^{(n)}\right)$ değerleri hesapla
3. $\delta^{(n+1)} = -f\left(x^{(n)}\right) / f'\left(x^{(n)}\right)$ formülünden birbirini takip eden aşamalarda bulunan kök ara değerleri arasındaki farkı hesapla
4. $x^{(n+1)} \leftarrow x^{(n)} + \delta^{(n+1)}$ formülünden de (n+1).nci aşamada kök için bulunan ara değeri hesapla
5. Eğer $\left|\delta^{(n+1)}\right| < \varepsilon$ ise, yakınsayan kök $\rightarrow x^{(n+1)}$ değerinde saklıdır. Adım (6)'a git. Kök bulunmamış ise, $\left|\delta^{(n+1)}\right| > \varepsilon$ işlem (2).nci adıma gönder, (2)-(5) adımlarını kök bulununcaya kadar devam ettir
6. Kökü ($x^{(n+1)}$) ekrana yazdır ve programı durdur.

ÖRNEK 1: $e^x = 3x$ eşitliğini sağlayan x değerini bulalım.

Bu eşitliği $f\left(x\right) = e^x - 3x = 0$, yani sıfıra eşitlenecek şeklinde yazıldığında $f\left(x\right)$'in kökü bu eşitliği sağlayacaktır.

Başlangıç tahmin değerini $x^{(0)} = 0.3$ alırsak ve yukarıda verilen algoritma uyarınca gerekli işlemleri yapacak olursak Tablo 12.1 'deki sayısal değerler elde edilir. Burada sadece altı basamak hassasiyetle çalışılmıştır.

```
PROGRAM kokbul
IMPLICIT NONE
REAL :: x, delta, x0=0.3, eps=1.e-5
REAL :: f, p
INTEGER :: n
!
!  NEWTON-RAPHSON YÖNTEMİ İLE KÖK BULMA İŞLEMİNİ
!  YAPAN PROGRAM.
!    * n    = İTERASYON SAYACI
!    * X0   = KÖK İÇİN BAŞLANGIÇ TAHMİNİ,
!    * EPS  = YAKINSAMA İÇİN KULLANILACAK KRİTER,
```

```
!    * F(X) = KÖKÜ ARANAN FONKSİYON
!    * P(X) = DF/DX, YANİ F'in X'e GÖRE TÜREVİDİR.
!
WRITE(6,50)
x=x0               ! Çözümü başlangıç değerine ata
n=0                ! İterasyon sayacını sıfırla
DO
   WRITE(6,60) n,x,f(x)
   n=n+1           ! n+1'ci iterasyon işlemlerini yap
   delta=f(x)/p(x)    ! δ^(n+1) = f(x^(n)) / f'(x^(n))
   x=x-delta          ! x^(n+1) ← x^(n) + δ^(n+1)
   IF(ABS(delta)<eps) EXIT   ! Yakınsadı! Dögüden çık!
END DO
WRITE(6,70) x
50 FORMAT(5X,'i',10X,'x',15X,'Fx',/,3X,35('-')/)
60 FORMAT(3X,i3,3X,f10.7,5X,f12.7)
70 FORMAT(//10X,'KöK=',f14.8//)
END PROGRAM kokbul

REAL FUNCTION f(x)
! ** Kökü aranan fonksiyonu f = f(x) burada tanımlayın
IMPLICIT NONE
REAL, INTENT(IN) :: x
  f=EXP(x)-3.*x
END FUNCTION f

REAL FUNCTION p(x)
! ** f(x) fonksiyonunun türevidir.   p(x) = df / dx
IMPLICIT NONE
REAL, INTENT(IN) :: x
  p=EXP(x)-3.
END FUNCTION p
```

Tablo 12.1'den de görüldüğü üzere kök üç iterasyonda yakınsamaktadır.

Tablo 12.1: $e^x - 3x = 0$ denklemini Newton-Raphson metodu ile çözüm aşamaları.

n	$x^{(n)}$	$f\left(x^{(n)}\right)$	$f'\left(x^{(n)}\right)$
0	0.3000000	0.4498588	-1.6501411
1	0.5726184	0.0550479	-1.2270968
2	0.6174787	0.0018109	-1.1457529
3	0.6190593	0.0000023	-1.1428199

12.2.2 VON MİSES METODU

Newton-Raphson metoduna benzeyen fakat türevi bir çok kez hesaplamaya gerek bırakmayan bir metod da *von Mises* metodudur. Bu metod da benzer şekilde bir ardışık tekrar metodudur. Fonksiyonun kökü, aşağıda verilen ve ardışık işlem gerektiren ifade ile hesaplanır.

Ardışık tekrar için kullanılması tavsiye edilen işlem:

$$x^{(n+1)} = x^{(n)} - \frac{f\left(x^{(n)}\right)}{f'\left(x^{(0)}\right)}$$

Bu ifadeden de görüldüğü üzere Newton-Raphson formülünden tek farkı paydadaki türevin sadece başlangıç tahmin değeri için hesaplanması gerektiğidir. Bu metodun algoritması da Newton-Raphson'un kinin hemen hemen aynıdır.

Tablo 12.2: Örnek 1'in von Mises metodu ile çözüm aşamaları.

n	$x^{(n)}$	$f'\left(x^{(n)}\right)$
0	0.3000	0.4498
1	0.5726	0.0551
2	0.6059	0.0152
3	0.6151	0.0045
4	0.6178	0.0014
5	0.6186	0.0005
6	0.6189	0.0002
7	0.6190	0.0001

Yukarıda çözülen problemi bu metod ile çözüp ve sayısal değerleri Tablo 12.2'de özetlediğimizde, problemin çözümü için yapılan ardışık işlem sayısının Newton-Raphson metodundan fazla olduğu görülür.

Polinomların, kompleks katsayılı polinomların ve daha nice lineer olmayan denklemlerin köklerini bulmak için geliştirilen yöntemlerin sayısı oldukça fazladır. Mühendislik ve teknolojik öneme sahip metotların en önemlilerine Nümerik Analiz derslerinde değinilecektir. Burada kapsanan metotlar şu anda temel bilimlerde karşılaşabileceğiniz problemleri çözecek yeterliliktedir.

ALIŞTIRMALAR

12.1 Fraunhofer differaksiyonunun incelenmesinde

$$\left(\frac{\sin\alpha}{\alpha}\right)^2 = \frac{1}{2}$$

denklemi ile karşılaşmaktayız. Bu denklemde α 'yı radyan olarak alarak verilen eşitliği sağlayan α çözümünü bulunuz.

12.2 Aşağıda verilen eşitlik bir helisel yay sisteminin hareketini tanımlamaktadır:

$$\tan\beta - \beta = C$$

Burada β ile verilen açıyı (a) $C = 0.01$ ve (b) $C = 0.001$ alarak hesaplayınız.

12.3 Aşağıda verilen fonksiyonun bir tane katlı kökünü bulunuz. Başlangıç tahmin değeri olarak $x^{(0)} = 1$ alınız

$$f(x) = x^2 - 2xe^{-x} + e^{-2x}$$

12.4 Bir devreden geçen akımın zamana göre değişimi

$$i = 5e^{-5t}\sin(\pi/4) + 5\sin(5t - \pi/4)$$

olarak verilmiştir. Akımın sıfır olduğu zamanı saniye cinsinden bulunuz.

12.5 Hidrojen sülfür gazının parçalanma derecesi *X*

$$\left(1 - PK^2\right)X^3 - 3X + 2 = 0$$

denklemi ile verilmektedir. Burada *K* denge sabiti ve *P* (atm) toplam basınçtır. 2000 Kelvin sıcaklığında *K*=0.608 olduğu bilindiğine göre, H_2S'in l atm basınçta parçalanma derecesini bulunuz, *X'*in alabileceği değerlerin 0 ile l arasında değişmesi gerektiğine dikkat ediniz. Bu aralık içinden ve dışından başlangıç tahmin değerleri kullanarak problemi çözünüz ve sonuçları irdeleyiniz.

12.6 Van der Waals durum denklemi

$$\left(P + \frac{a}{v^2}\right)(v - b) = RT$$

ile verilmektedir. Bu denklemde *P* basınç, *v* molar hacim, *T* sıcaklık ve *R* gaz sabitini temsil etmektedir. Sabitlerden a, bağlanma kuvvetinin bir ölçüsü ve *b* de bir hacim düzeltme faktörü olarak kullanılmaktadır. Basınç *atm*, hacim *lt* ve sıcaklık *K* (kelvin) birimi olarak alındığında .*R*'nin değeri 0.082 lt-atm/K-mol'dür. Toluen için a ve *b* sabitleri

$$a = 24.06 \text{ lt-atm/mol} \qquad b = 0.1463 \text{ lt/mol}$$

olarak veriliyor. Toluen'in l atmosfer basınçta kaynama sıcaklığı olan 110°C sıcaklığındaki molar hacmini hesaplayınız.

12.7 Üçüncü dereceden denklemlerin köklerinin bulunması ile ilgili olarak verilen algoritmaya dayanarak, bu tür denklemlerin köklerini bulan bir genel program yazınız ve köklerini bildiğiniz denklemler ile kontrol ediniz.

12.8 Aşağıda verilen iki bilinmeyenli iki lineer olmayan denklemin köklerini Newton-Raphson metodunu uygulayarak çözmek istiyoruz.

$$a^5 + b^5 = 21 \qquad a^4 - b^4 = 11$$

Program yazmak için ilk önce bu sistem için Newton-Raphson formülünü geliştiriniz, sonra formülleri programlayınız (bilinmeyenleri 10^{-4}'ten küçük hata ile hesaplamak istiyoruz). VERİ: İki değişkenli fonksiyonlar için Taylor formülü:

$$f(x, y) = f(x_0, y_0) + (x - x_0)\left(\frac{\partial f}{\partial x}\right)_{(x_0, y_0)} + (y - y_0)\left(\frac{\partial f}{\partial y}\right)_{(x_0, y_0)}$$

NOT: Problem, bilinmeyen sayısı teke indirgenerek de çözülebilir.

12.9 Aşağıdaki polinomların tüm köklerini bulunuz.

(a) $2x^5 - 5x^4 + 4x - 1 = 0$
(b) $x^6 + x^5 - 2x^4 + 3x^3 + x^2 + 10x - 4 = 0$
(c) $x^7 + 5x^6 + 7x^5 + 8x^4 - 12x^3 + 49x - 54 = 0$
(d) $x^{10} + 8x^8 - 27x^7 - 28x^4 + 5x^3 - 9x^2 - 81x + 115 = 0$

12.10 Kalınlığı 10 cm olan ve oldukça yüksek bir plakanın sıcaklığı 100° C 'ta homojen olarak tutulmaktayken, t = 0 anından itibaren plakanın iki tarafından hava üflenerek soğutulmaya başlanmaktadır. Bu plakanın herhangi bir x noktasında ve t anındaki sıcaklığı $T(x,t)$

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2},$$

$$T(x,0) = 100°C, \qquad \frac{\partial T}{\partial t} = T(0,t), \qquad t > 0 \qquad \frac{\partial T}{\partial t} = -T(0.1,t), \qquad t > 0$$

diferansiyel denklemi ve sınır şartlarıyla beraber verilmektedir. Bu kısmi diferansiyel denklemin analitik çözümü aşağıda verilen seri şeklinde edilmiştir:

$$T(x,t) = 400\sum_{n=1}^{\infty}\left\{\frac{\sec\alpha_n}{\left(3+4\alpha_n^2\right)}\exp\left[-4\alpha_n^2 t\right]\cos\left(2\alpha_n\left(x-\frac{1}{2}\right)\right)\right\}, \qquad 0 \le x \le 0.1m$$

Burada α_n değerleri

$$\alpha \tan\alpha = \frac{1}{2}$$

denkleminin pozitif kökleridir. Bu seri ifadesinden faydalanarak plakanın herhangi bir x (metre) noktasının t (saniye) anındaki sıcaklığını bulan bir program yazınız. *NOT:* Önce α değerlerini bulunuz ve bunları programa bir `MODULE` ilave ediniz ve `USE` deyimi ile ilgili programlarda yararlanınız.

BÖLÜM 13
MATRİSLER VE VEKTÖRLER

Matrisler ve vektörler ile uğraşırken indisli değişkenler kullanılmak zorundadır. Matrisler ile uğraşmamızın nedeni ise birçok problemin çözümünde lineer denklem sistemlerinin karşımıza çıkmasıdır. N bilinmeyenli N denklemin çözümü için yapılması gerekli işlerden birinin de matrisin tersini hesaplamak olduğudur.

Lineer denklem sistemleri diferansiyel denklemlerin nümerik çözümleri, yapısal analiz, devre analizi, optimizasyon, ve veri analizi gibi birçok problemlerin çözümlerinde karşımıza çıkarlar. Oldukça fazla sayıda denklemden oluşan sistemlerin çözümlerine sıkça karşılaşılır. Bu durumda denklem sistemlerini hem doğru hem de verimli bir şekilde çözmek için uygun bir algoritma saptamak önemli olmaktadır.

13.1 TEMEL BİLGİLER

Bir matrisin n satır ve n sütunu var ise bu matrise *Kare Matris* denir ve $n \times n$ ile gösterilir. Örnek olarak 4×4 'lük bir matris aşağıdaki şekilde verilmiş olsun.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Köşegen a_{11}, a_{22}, a_{33} ve a_{44} elemanlarından oluşmakta olup, bu terimlere matrisin *Ana Köşegeni* denir. Eğer matrisin elemanları arasından $a_{ij} = a_{ji}$ bağıntısı var ise matris *simetriktir* denir. Bir *Üst Üçgensel Matris* ana köşegen altında kalan bütün elemanları sıfır olan matristir.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{pmatrix}$$

Yukarıda verilen **A** matrisi bir üst üçgensel matristir. Benzer şekilde *Alt Üçgensel Matris* de matrisin ana köşegeni üstünde kalan bütün elemanları sıfır olan matris olarak tanımlanır.

Birim matris **I** ile gösterilir ve ana köşegeni "1" diğer elemanları sıfır olan matristir. *Band Matris* ise ana köşegen etrafındaki köşegenlerde sıfır olmayan elemanlardan yoğunlaşmış matristir.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & a_{43} & a_{44} \end{pmatrix}$$

A matrisi band genişliği üç olan bir band matristir ve band genişliği üç olan matrislere ayrıca *Üç Köşegenli matris* de denir.

A, **B** ve **C** $n \times n$'lik kare matrislerinin elemanları sırasıyla a_{ij}, b_{ij} ve c_{ij} olarak verilmiş olsun. $\mathbf{C} = \mathbf{A} \pm \mathbf{B}$ operasyonu neticesinde **C** matrisinin elemanları $c_{ij} = a_{ij} \pm b_{ij}$ ifadesi ile hesaplanır. Eğer **C** matrisi bir çarpım matrisi ise, yani, $\mathbf{C} = \mathbf{A} \times \mathbf{B}$, o zaman çarpım matrisinin elemanları:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

ifadesinden hesaplanır. Program 1'de bu işlemi yapan bir `SUBROUTINE` alt programı verilmektedir. Eğer **A** matrisi, boyutu *n* olan bir **X** vektörü ile sağ taraftan çarpılıyorsa, sonuç bir vektördür. Bu sonuç vektöre **R** dersek, yani, $\mathbf{R} = \mathbf{A} \times \mathbf{X}$, çarpım vektörünün elemanları,

$$r_i = \sum_{k=1}^{n} a_{ik} x_k \tag{13.1}$$

ifadesinden hesaplanabilir. Bu işlemi gerçekleştiren bir `SUBROUTINE` alt programı da Program 2'de verilmiştir.

Dördüncü mertebeden bir lineer denklem sistemin (4 x 4) matris formundaki eşdeğeri

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{pmatrix} \tag{13.2}$$

veya aşağıdaki şekilde bir matris denklemi ile ifade etmek mümkündür.

$$\mathbf{AX} = \mathbf{R}$$

Burada **A** matrisi 4×4'lük bir kare matris, **X** uzunluğu 4 olan bilinmeyenlerden oluşan vektör ve **R**'de aynı uzunlukta olan sağ-taraf vektörüdür.

Lineer denklem sistemlerinin çözümünde kullanılan metodlardan birisi de Kramer kuralıdır ve herhangi bir x_i için çözüm ile verilir. Burada $\det \mathbf{A}_i$, matrisin *i*'.nci sütunu **R** vektörü ile değiştirilmiş matrisin determinantını ifade etmektedir. Lineer denklem sistemini ($n \times n$ için) Kramer metodu ile çözmek için yapılması gereken temel aritmetik işlem. sayısı n^4 ile orantılıdır. Bu metodun bilgisayarlarda lineer denklem sistemini çözmede kullanılması, gerektirdiği işlem sayısı itibariyle pratik değildir; bu nedenle bilgisayarda daha az işlem gerektiren ve daha pratik olan metotlara değineceğiz.

ÖRNEK 1: İki matrisin çarpımını yapan bir alt program hazırlayınız, ana program kullanarak test ediniz.

```
PROGRAM Ornek1
IMPLICIT NONE
REAL, DIMENSION(3,3) :: c, b
REAL, DIMENSION(3,3) :: a=RESHAPE( (/ 1., 2., 3., &
              4., 5., 6., 7., 8., 9. /), (/3,3/))
INTEGER :: i,j,n=3
b=RESHAPE( (/ 2.,-2., 3., 1., 0., 4., 1., 2., &
              -7. /),(/3,3/) )
WRITE(*,20) 'A'
DO i=1,n
   WRITE(*,10) (a(i,j),j=1,n)
END DO
WRITE(*,20) 'B'
DO i=1,n
   WRITE(*,10) (b(i,j),j=1,n)
END DO
CALL mcarp(n,a,b,c)  ! Matris çarpımı yapıldı
WRITE(*,20) 'C'
DO i=1,n
   WRITE(*,10) (c(i,j),j=1,n)
END DO
10 FORMAT(5(2x,F7.1))
20 FORMAT(2x,A,' matrisi')
END PROGRAM Ornek1

SUBROUTINE mcarp(n,a,b,c)
IMPLICIT NONE
INTEGER, INTENT(IN):: n
REAL, DIMENSION(n,n), INTENT(IN)  :: a
REAL, DIMENSION(n,n), INTENT(IN)  :: b
REAL, DIMENSION(n,n), INTENT(OUT) :: c
INTEGER :: i,j,k
!
! C=AxB ISLEMINI YAPAN ALTPROGRAM
!
! ** A (nxn) BOYUTLU MATRISTIR
! ** B (nxn) BOYUTLU MATRISTIR
! ** C (nxn) BOYUTLU MATRISTIR (=A*B)
DO i=1,n
   DO j=1,n
      c(i,j)=0.0
       DO k=1,n
          c(i,j)=c(i,j)+a(i,k)*b(k,j)
       END DO
   END DO
END DO
END SUBROUTINE mcarp
```

Programın çıktısı

```
A matrisi
    1.0      4.0      7.0
    2.0      5.0      8.0
    3.0      6.0      9.0
B matrisi
    2.0      1.0      1.0
   -2.0      0.0      2.0
    3.0      4.0     -7.0
C matrisi
   15.0     29.0    -40.0
   18.0     34.0    -44.0
   21.0     39.0    -48.0
```



Normalde matris çarpımını programlamanıza gerek yoktur; arşiv programı olan **`MATMUL(A,B)`***bu işlevi görmektedir.* **`A,B`** *boyutları uyumlu çarpım sırasına göre verilen matrislerdir.*

ÖRNEK 2: Bir matris ile vektörün çarpımına ilişkin alt program hazırlayınız ve test ediniz.

Alt programda Denklem (13.1)'in programlanması gerekmektedir.

```
PROGRAM Ornek2
IMPLICIT NONE
REAL, DIMENSION(3) :: x=(/ 2., 4., -7. /), r
REAL, DIMENSION(3,3) :: a=RESHAPE( (/ 1., 2., 3., &
               4., 5., 6., 7., 8., 9. /), (/3,3/))
INTEGER :: i,j, n=3
!    Matrisleri yazdır
WRITE(3,20) 'A'
DO i=1,n
   WRITE(3,10) (a(i,j),j=1,n)
END DO
WRITE(3,20) 'x'
DO i=1,n; WRITE(3,10) x(i); END DO
CALL vcarp(n,a,x,r)
WRITE(3,20) 'r'
DO i=1,n; WRITE(3,10) r(i); END DO
10 FORMAT(5(2x,F7.1))
20 FORMAT(/2x,A,' matrisi')
END PROGRAM Ornek2

SUBROUTINE vcarp(n,a,x,r)
IMPLICIT NONE
INTEGER, INTENT(IN) :: n
REAL, DIMENSION(n),  INTENT(IN) :: x
REAL, DIMENSION(n,n),INTENT(IN) :: a
REAL, DIMENSION(n),  INTENT(OUT):: r
```

```
INTEGER :: i, k
! R=AxX ISLEMINI YAPAN ALTPROGRAM
!
! ** a (nxn) BOYUTLU MATRISTIR
! ** x (1:n) BOYUTLU VEKTÖRDÜR
! ** r (1:n) BOYUTLU VEKTÖRDÜR (=a*x)
DO  i=1,n
    r(i)=0.0
    DO  k=1,n
        r(i)=r(i)+a(i,k)*x(k)
    END DO
END DO
END SUBROUTINE vcarp
```

Programının çıktısı

```
  A matrisi
      1.0      4.0      7.0
      2.0      5.0      8.0
      3.0      6.0      9.0

  x matrisi
      2.0
      4.0
     -7.0

  r matrisi
    -31.0
    -32.0
    -33.0
```

Matris denklemi şeklinde verilen ifadeden hareketle eşitliğin iki tarafı **A** matrisinin tersi ile soldan çarpılınca çözümü, matris notasyonu ile,

$$\mathbf{X} = \mathbf{A}^{-1}\mathbf{R}$$

şeklinde elde edilebilir. Fakat bu denklem sisteminin çözümü matrisin tersini hesaplamaya gerek kalmadan da yapılabilir ve bu metodlar bilgisayar zamanı açısından da daha verimli metodlardır.



RESHAPE *niteliği, bir boyutlu bir matrisin* $\boldsymbol{n}\times\boldsymbol{m}$ *boyutlu bir matris olarak ifade edilmesi için kullanılan bir deyimdir. Kullanım şekli;*
REHAPE(**(/1,2,..veriler /)**, **(/n,m/)**) *olarak verilir.*

13.2 YOKETME METODLARI

Lineer denklem sistemlerinin çözümleri ve ters matrisin bulunması işlemi için en sık kullanılan doğrudan çözüm metotlardır. Bu metotlar oldukça eski olmalarına rağmen sayısal hesaplarda en etkili ve pratik metotlardır.

Aşağıda verilen dört bilinmeyenli dört denklemin matris formunu, Denklem (13.2)'i ele alalım.

Gauss yoketme metodu aslında satır işlemlerinden ibarettir. Önce birinci satır a_{11} ile bölünür:

$$\begin{pmatrix} 1 & a'_{12} & a'_{13} & a'_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} r'_1 \\ r_2 \\ r_3 \\ r_4 \end{pmatrix}$$

Burada (') ile gösterilen matris elemanları bu elemanların değiştiğini ifade etmektedir. Birinci denklem a_{21} ile çarpılıp ikinci denklemden çıkartılır.

$$\begin{pmatrix} 1 & a'_{12} & a'_{13} & a'_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} r'_1 \\ r'_2 \\ r_3 \\ r_4 \end{pmatrix}$$

Birinci denklem şimdi a_{31} ile çarpılıp üçüncü denklemden daha sonra da a_{41} ile çarpılıp dördüncü denklemden çıkartılırsa,

$$\begin{pmatrix} 1 & a'_{12} & a'_{13} & a'_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & a'_{32} & a'_{33} & a'_{34} \\ 0 & a'_{42} & a'_{43} & a'_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} r'_1 \\ r'_2 \\ r'_3 \\ r'_4 \end{pmatrix}$$

elde edilir.

Aynı şekilde ikinci satır a'_{22} ile bölünür ve bu elemanın bulunduğu sütunda ve altındaki elemanlar benzer şekilde sıfırlanır. Böylece köşegen üstündeki elemanlar "1" yapılırken alt üçgensel kısım sıfırlanmış olur ve sonuç olarak denklem sistemimiz aşağıdaki şekli alır.

$$\begin{pmatrix} 1 & a'_{12} & a'_{13} & a'_{14} \\ 0 & 1 & a'_{23} & a'_{24} \\ 0 & 0 & 1 & a'_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} r'_1 \\ r'_2 \\ r'_3 \\ r'_4 \end{pmatrix}$$

Denklem (13.1) ile verilen sistemimizi bilinmeyenler için çözülebilecek bir sekile sokmuş oluyoruz. Gauss yoketme metodunda bilinmeyenleri en alttaki denklemden başlayarak en üstteki denkleme doğru ilerleyerek buluyoruz. En alttaki denklemden,

$$x_4 = r'_4$$

üçüncü denklemden,

$$x_3 + a'_{34}x_4 = r'_3 \quad \text{veya} \quad x_3 = r'_3 - a'_{34}x_4$$

ikinci denklemden,

$$x_2 + a'_{23}x_3 + a'_{24}x_4 = r'_2 \quad \text{veya} \quad x_2 = r'_2 - a'_{23}x_3 - a'_{24}x_4$$

son olarak da birinci denklemden,

$$x_1 + a'_{12}x_2 + a'_{13}x_3 + a'_{14}x_4 = r'_1 \quad \text{veya} \quad x_1 = r'_1 - a'_{12}x_2 - a'_{13}x_3 - a'_{14}x_4$$

bulunur.

Gauss-Jordan metodunda ise yoketme işlemi üst üçgensel kısım için de uygulanır. Bu durumda,

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} r_1'' \\ r_2'' \\ r_3'' \\ r_4'' \end{pmatrix}$$

nihai denklem sistemi elde edilir. Buradan denklem sisteminin çözümünün sağ taraf vektörü, yani $x_1 = r_1''$, $x_2 = r_2''$, $x_3 = r_3''$, ve $x_4 = r_4''$ olduğu görülür.

Gauss yoketme metodu için bir algoritma aşağıdaki şekilde verilmektedir. Bu algoritma da kullanılan $i = a(b)c$ şeklindeki gösterimin anlamı $i = a$ 'dan *c*'ye kadar *b*'lik artırımları kastetmektedir. Fortran 90/95 programlama dilinde **`DO i=a,c,b`** deyimine karşılık gelmektedir.

1. **A** kare matrisi ve **R** sağ taraf vektörü temin edilir.
2. $k = 1(1)n$ 'e kadar $p = 1/a_{kk}$ ve $r_k = r_k \times p$ hesaplandıktan sonra 3-6. aşamalar her k için tekrarlanır.
3. $i = (k+1)(1)n$ için $a_{ki} = a_{ki} \times p$ hesaplanır.
4. $i = (k+1)(1)n$ 'e kadar 5-6 aşamalar her *i* için tekrarlanır.
5. $j = (k+1)(1)n$ 'e kadar Gauss yoketme işlemi matris için $a_{ij} = a_{ij} - a_{ik} \times a_{kj}$ ifadesinden hesaplanır.
6. Sağ taraf vektörü ise yoketme sırasında $r_i = r_i - a_{ik} \times r_k$ ifadesi gereğince değiştirilir.
7. Yoketme işlemi tamamlandıktan sonra x_n 'nin çözümü $x_n = r_n$ dir.
8. $k = (n-1)(-1)1$ 'e kadar 9-11. aşamalar her *k* için tekrarlanır.
9. $Toplam = 0$ alınır.
10. $j = (k+1)(1)n$ 'e kadar $Toplam = Toplam + a_{kj} \times x_j$ ifadesi hesaplanır.
11. *k*.ncı bilinmeyen $x_k = r_k - Toplam$ formülünden hesaplanır.

ÖRNEK 3: Gauss yoketme metodu ile *n* bilinmeyenli *n* denklem sisteminin çözümünü yapan altprogram aşağıda verilmiştir.

```
SUBROUTINE Gauss_Yoketme_Metodu(n,a,r,x)
IMPLICIT NONE
INTEGER, INTENT(IN) :: n
REAL, DIMENSION(n), INTENT(INOUT)  :: r, x
REAL, DIMENSION(n,n), INTENT(INOUT):: a
REAL    :: toplam, p
INTEGER :: i, j, k
!
!  Gauss Yoketme Metodu ile Lineer Denklem Sisteminin
!  Çözümünü verir.
!  (n x n)   ŞEKLİNDEKİ LİNEER DENKLEM SİSTEMİNİN
```

```
!   ** a(n,n) : DENKLEM SİSTEMİNİ BELİRLEYEN MATRİS
!   ** r(n)   : SAĞ TARAF VEKTÖRÜ
!   ** x(n)   : BİLİNMEYENLER VEKTÖRÜDÜR.
!  UYARI! Alt programa girilen matris ve vektörlerin
!         başlangıç değerleri bozulmaktadır...
DO k=1,n
   p=1.0/a(k,k)
   r(k)=r(k)*p
    DO i=k+1,n
       a(k,i)=a(k,i)*p
    END DO
    DO i=k+1,n
       DO j=k+1,n
          a(i,j)=a(i,j)-a(i,k)*a(k,j)
       END DO
       r(i)=r(i)-a(i,k)*r(k)
    END DO
END DO
! Sondan başa doğru çözümlerin elde edilmesi
x(n)=r(n)

DO k=n-1,1,-1
   toplam=0.0
   DO j=k+1,n
      toplam=toplam+a(k,j)*x(j)
   END DO
   x(k)=r(k)-toplam
END DO
END SUBROUTINE Gauss_Yoketme_Metodu
```

Gauss-Jordan yoketme metodu için hazırlanabilecek herhangi bir algoritma Gauss yoketme metodun da gerçekleştirilen alt üçgensel matrisin sıfırlanması işlemini içerecektir. Bu nedenle algoritma da Gauss yoketme yöntemi için verilen algoritmanın sadece 7-11.ci aşamalarının aşağıdaki şekilde değiştirilmesi yeterli olacaktır.

7. $j = n(-1)2$ 'e kadar 8.nci aşama tekrarlanır.
8. Üst üçgensel matris sıfırlanırken sağ taraf vektörü $i = (j-1)(-1)1$'e kadar $r_i = r_i - a_{ij} \times r_j$ şeklinde değiştirilir.
9. Çözüm vektörü sağ taraf vektörüne eşittir ve $j = 1(1)n$ 'e kadar $x_i = r_i$, alınır.

Gauss-Jordan metodu için hazırlanan alt program Gauss yoketme metodu için verilen alt programda algoritma gereği yapılan değişiklikler burada verilmektedir.

```
. . .
END DO
! aşağıdaki kısım ilave edilecek
DO j=n,2,-1
   DO i=(j-1),1,-1
      r(i)=r(i)-a(i,j)*r(j)
```

```
   END DO
END DO
DO i=1,n
   x(i)=r(i)
END DO
END SUBROUTINE Gauss_Yoketme_Metodu
```

Örneğin aşağıdaki denklem sistemini `Gauss_Yoketme_Metodu` alt programını kullanarak çözen bir ana program oluşturmak istersek,

$$\begin{pmatrix} 2 & -1 & 3 \\ 1 & 5 & -2 \\ 4 & 2 & -3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 14 \\ -9 \\ -3 \end{pmatrix}$$

```
PROGRAM Ornek3
IMPLICIT NONE
REAL, DIMENSION(3) :: r=(/ 14., -9., -3. /), x, rr
REAL, DIMENSION(3,3) :: a=RESHAPE( (/ 2., 1., 4., &
    -1., 5., 2., 3., -2., -3. /), (/3,3/))
INTEGER :: i,j, n=3
OPEN(3,FILE='Cikti.dat',STATUS='unknown')
!
rr=r                 ! Alt program r'nin değerlerini bozacaktır
CALL Gauss_Yoketme_Metodu(n,a,r,x)
DO i=1,n
   WRITE(3,10) (a(i,j),j=1,n),i,rr(i)
END DO
WRITE(3,15)
DO i=1,n
   WRITE(3,20) i,x(i)
END DO
10 FORMAT(2x,'[',3(2x,F5.1),'] [ x(',i1, &
          ') ]= [',F5.1,']')
15 FORMAT(/2x,'Denklem Sisteminin Çözümü'/)
20 FORMAT(2x,' x(',i2,') = ',F5.2)
END PROGRAM Ornek3
```

Programının çıktısı

```
[    2.0   -0.5    1.5] [ x(1) ]= [ 14.0]
[    1.0    5.5   -0.6] [ x(2) ]= [ -9.0]
[    4.0    4.0   -6.5] [ x(3) ]= [ -3.0]

Denklem Sisteminin Çözümü

x( 1) =  2.00
x( 2) = -1.00
x( 3) =  3.00
```

13.3 TERS MATRİS HESABI

Bir **A** kare matrisinin tersi eğer matris tekil değilse mevcuttur; yani, $\det \mathbf{A} \neq 0$. Matrisin tersini bulmak aynı lineer denklem sisteminin çözümüne benzemektedir. Buradaki tek fark sağ taraf vektörü yerine sağ tarafa birim matrisi yerleştirerek Gauss-Jordan yoketme metodunun uygulanmasıdır. İşlem tamamlandığında sol tarafta birim matris oluşurken sağ tarafta (yani, birim matrisin bulunduğu yerde) matrisin tersi oluşur. Bu işlemleri açıklamak için **A** kare matrisini aynı boyutlu birim matris ile beraber aşağıdaki şekilde yazalım.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Şimdi Gauss-Jordan yoketme işlemine başlayalım. İlk önce birinci satırı a_{11} ile bölelim.

$$\begin{pmatrix} 1 & a'_{12} & a'_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \begin{pmatrix} b_{11} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Daha sonra birinci sütundaki matrisin diğer elemanlarını sıfırlamak için birinci satırı sıfırlanacak satır elemanı ile çarpıp o satırdan çıkartalım.

$$\begin{pmatrix} 1 & a'_{12} & a'_{13} & a_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & a'_{32} & a'_{33} & a'_{34} \\ 0 & a'_{42} & a'_{43} & a'_{44} \end{pmatrix} \begin{pmatrix} b_{11} & 0 & 0 & 0 \\ b_{21} & 1 & 0 & 0 \\ b_{31} & 0 & 1 & 0 \\ b_{41} & 0 & 0 & 1 \end{pmatrix}$$

Bu işlem yapılır iken birim matrisin bulunduğu kısım da yoketme işleminin uygulandığı sütundaki elemanlar değişmektedir. Yok etme işlemi soldaki matris birim matris'e dönüşünceye kadar devam ettirildiğinde

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{pmatrix}$$

elde edilir. Sağ tarafta oluşan ve elemanları b_{ij} ile verilen kare matris artık **A** matrisinin tersini oluşturmaktadır.

Gauss- Jordan metodu ile ters matrisin bulunması için bir algoritma verilmektedir.

1. Programa **A** matrisi temin edilir.
2. **B** birim matrisi $i = 1(1)n$'e kadar $b_{ii} = 1$ve $b_{ij} = 0$ alınarak oluşturulur.
3. $j = 1(1)n$'e kadar $p = 1/a_{jj}$ alındıktan sonra her j için 4-6.ncı aşamalar tekrarlanır.

4. **A** matrisinin köşegenleri "1" yapılırken birim matris ve ana matrisin *j*'.nci satırlarındaki elemanlar $k = 1(1)n$'e kadar

   $b_{jk} = b_{jk} \times p$ ve $a_{jk} = a_{jk} \times p$ şeklinde değiştirilir.
5. $i = 1(1)n$'e kadar $T = a_{ij}$ alındıktan sonra 6.ncı aşama her *i* için tekrarlanır.
6. *j*.nci satırda yoketme işlemi uygulanırken hem **A** hem de birim matrisin elemanları $k = 1(1)n$'e kadar

   $b_{ik} = b_{ik} - T \times b_{jk}$ ve $a_{ik} = a_{ik} - T \times a_{jk}$ şeklinde değiştirilir.

ÖRNEK 4: *n*.nci mertebeden bir kare matrisinin tersini Gauss-Jordan metodu ile hesaplayan altprogram.

```
SUBROUTINE Jordan(n,a,b)
IMPLICIT NONE
INTEGER, INTENT(IN) :: n
REAL, DIMENSION(n,n), INTENT(INOUT) :: a
REAL, DIMENSION(n,n), INTENT(OUT)   :: b
INTEGER :: i, j, k
REAL    :: p, t
!   Bu alt program (nxn)'lik bir kare matrisin
!   Gauss-Jordan metodu ile matrisin tersini hesaplar
!
!   ** n  : Kare matrisin kenar boyutu       (Girdi)
!   ** a  : Tersi aranan nxn'lik kare matris (Girdi)
!   ** b  : nxn'lik matris a'nın tersi       (Çıktı)
!
!   UYARI! Bu program çıkışında A matrisi ve R sağ taraf vektörünün
!          girişteki değerleri değişmektedir
!
DO i=1,n
   b(i,i)=1.0      ! Birim matris oluşturuldu
END DO
DO j=1,n
   p=1.0/a(j,j)
   DO k=1,n
      b(j,k)=b(j,k)*p
      a(j,k)=a(j,k)*p
   END DO
   DO i=1,n
      t=a(i,j)
      IF(i==j) CYCLE
      DO k=1,n
         b(i,k)=b(i,k)-t*b(j,k)
         a(i,k)=a(i,k)-t*a(j,k)
      END DO
   END DO
END DO
END SUBROUTINE Jordan
```

ÖRNEK 5: Aşağıda verilen **A** matrisinin tersini ($\mathbf{A}^{-1}$) Gauss-Jordan yöntemiyle hesaplayan ve Örnek 4'de verilen `Jordan` alt programını kullanan bir ana program yazınız. Program $\mathbf{A}^{-1}\mathbf{A}=\mathbf{I}$ denklemini sağlayıp sağlamadığını kontrol etmesi istenmektedir.

$$\mathbf{A} = \begin{pmatrix} 1 & -2 & -2 \\ -2 & 1 & 1 \\ 2 & 1 & -1 \end{pmatrix}$$

Daha önce verilen Gauss-Jordan algoritmasına göre hazırlanan `Jordan` isimli alt programını kullanan ana program aşağıda verilmiştir.

```
PROGRAM Ornek5
IMPLICIT NONE
REAL, DIMENSION(3,3) :: a=RESHAPE( (/ 1., -2., 2., &
-2., 1., 1., -2., 1., -1. /), (/3,3/)), birim, aa, c
INTEGER :: i,j, n=3
OPEN(3,FILE='Cikti.dat',STATUS='unknown')
!
WRITE(3,20) 'A'
DO i=1,n
   WRITE(3,10) (a(i,j),j=1,n)
END DO
aa=a !A'nın içeriği değiştiğinden sağlama amacıyla aa matrisinde saklanıyor
!
CALL Jordan(n,a,birim)
!
WRITE(3,20) 'Tersi'
DO i=1,n
   WRITE(3,10) (birim(i,j),j=1,n)
END DO
CALL mcarp(n,birim,aa,c)  ! Sağlama yapılıyor
WRITE(3,20) 'A.A^(-1)'
DO i=1,n
   WRITE(3,10) (c(i,j),j=1,n)
END DO
10 FORMAT(5(2x,F9.5))
20 FORMAT(/2x,A,' matrisi')
END PROGRAM Ornek5
```

Programının çıktısı aşağıda verilmiştir

```
  A matrisi
    1.00000   -2.00000   -2.00000
   -2.00000    1.00000    1.00000
    2.00000    1.00000   -1.00000

  Tersi matrisi
   -0.33333   -0.66667    0.00000
    0.00000    0.50000    0.50000
   -0.66667   -0.83333   -0.50000
```

```
A.A^(-1) matrisi
  1.00000    0.00000    0.00000
  0.00000    1.00000    0.00000
  0.00000    0.00000    1.00000
```

Görüldüğü üzere matris ile tersinin çarpımı birim matrisi vermektedir; yani bulunan ters matris hesabı doğru yapılmıştır.

Mühendisler, hesaplarında çoğu kez matrislerle ve özellikle lineer denklem sistemlerinin çözümleri ile uğraşırlar. Bu matrislerin boyutları onlar, yüzler ve hatta binlerle ifade edilebilir. Bu matrisleri bilgisayara elden veya kütükten girmek oldukça zor ve zahmetlidir. Ancak matrislerin elemanları, çoğu kez matriste bulundukları satır ve sütunun indisleri ile, örneğin, $a_{i,j} = 10 + (i^2 + j^2)/200$ gibi, belirli bir bağıntıya tabidirler. Bu bağıntıların bilinmesiyle, kısa bir program parçasıyla, arzulanan boyutta matrisler kolaylıkla oluşturulabilir.

ÖRNEK 6: Aşağıda verilen matrisi, dışarıdan girilen herhangi bir *n* değeri için oluşturan ve ekrana yazdıran bir program yazınız.

$$\begin{bmatrix} 1 & 1/2 & 1/3 & \cdots & 1/n \\ 0 & 1/2 & 1/3 & \cdots & 1/n \\ 0 & 0 & 1/3 & \cdots & 1/n \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 1/n \end{bmatrix}_{n\times n}$$

Bu matrisi oluştururken yapılacak ilk ve en önemli şey, matrisin elemanları ile bulundukları satır ve sütun indisleri arasındaki ilişkiyi tespit etmektir. Gözlem ile bu ilişkinin

$$a_{i,j} = \begin{cases} 1/j, & j \geq i \\ 0, & j < i \end{cases}$$

olduğu görülmektedir. Bu bağıntıya göre programı aşağıdaki şekilde oluşturabiliriz.

```
PROGRAM Ornek6
IMPLICIT NONE
INTEGER, PARAMETER :: n=5
REAL, DIMENSION(n,n) :: a
INTEGER :: i, j
! Matrisi oluştur
DO i=1,n
   DO j=1,n
      IF(j>=i) THEN
         a(i,j)=1.0/REAL(j)
      ELSE
         a(i,j)=0.0
      END IF
   END DO
END DO
! Matrisi yazdır
```

```
DO i=1,n
   WRITE(*,10) (a(i,j),j=1,n)
END DO
10 FORMAT(10(1x,F6.3))
END PROGRAM Ornek6
```

Program çıktısı

```
 1.000  0.500  0.333  0.250  0.200
 0.000  0.500  0.333  0.250  0.200
 0.000  0.000  0.333  0.250  0.200
 0.000  0.000  0.000  0.250  0.200
 0.000  0.000  0.000  0.000  0.200
```

olarak elde edilir.

13.4 BİR BOYUTLU MATRİSLER

İndisli değişkenler kullanırken, gerek tek boyutlu gerekse daha fazla boyutta, gereksiz bellek kullanımına gidilmemelidir. Örneğin bir programda kullanılacak indisli değişken boyutu, `DIMENSION` deyiminde bildirilen, maksimum sınırı aşmamalıdır.

İki ve daha fazla indisli değişkenlerde, örneğin matrislerin tanımında, her hangi bir (*i*,*j*) değerine ulaşmak tek indisli bir (*i*) değerine ulaşmaktan daha fazla zaman gerektirir; bu durum bilgisayarın çok indisli değişkenleri bellekte tek indisli değişken gibi saklı tutmasından kaynaklanmaktadır; yani $n \times m$ boyutlu bir **A** matrisinin elemanları bellekte $a_{11}, a_{21}, \ldots, a_{n1}$, $a_{12}, a_{22}, \ldots, a_{n2}$, $\ldots$, $a_{1m}, a_{2m}, \ldots, a_{nm}$ sırasıyla saklanmaktadır. Matrisin herhangi bir elemanına erişmek için bu tek sıranın nerede olduğu bulunduktan sonra o sıradaki bilgiye erişilir.

Bu nedenle biz de matrisi $n \times m$ boyutunda tek indisli bir değişken olarak atarsak; yani $a_1, a_2, a_3, \ldots, a_k$, bu durumda $k = i + (j-1)m$ şeklinde alındığı takdirde herhangi bir *i*, *j* çiftiyle tanımlanan matris elemanı a_k 'yı verir. Burada *k* ile verilen ifade matrisin bellekteki diziliş sırasını vermektedir.

Aşağıda boyutu $n \times n$ olan iki matris (A ve B)'nin programa okutulması ve çarpımını ($\mathbf{A} \times \mathbf{B}$) içeren bir program tek indisli değişkenler cinsinden verilmektedir.

```
PROGRAM mcarp
IMPLICIT NONE
!  A,B Matrisini kütükten Oku N'in Max Değeri 50 olabilir.
REAL, DIMENSION(2500) :: a, b, c
INTEGER :: i,j,k,n,n2,k1,k2,k3
READ(5,*) n
n2=n*n
! A ve B matrislerini Oku
READ(5,*) (a(k),k=1,n2)
READ(5,*) (b(k),k=1,n2)
! C = A x B'yi hesapla
```

```
DO i=1,n
   DO j=1,n
      k1=i+(j-1)*n
      c(k1)=0.0
      DO k=1,n
         k2=i+(k-1)*n
         k3=k+(j-1)*n
         c(k1)=c(k1)+a(k2)*b(k3)
      END DO
   END DO
END DO
! C'yi ekrana yaz
DO i=1,n
   WRITE(6,*) (c(i+(j-1)*n),j=1,n)
END DO
END PROGRAM mcarp
```

ALIŞTIRMALAR

13.1 Aşağıdaki şekilde verilen Üç Köşegenli matris sistemini Gauss yoketme tekniği ile çözmek için (i) bir Algoritma hazırlayınız (Bu Algoritma sıfırlı elemanlarla işlem yapmasın) (ii) Bir Fortran alt programı yazınız.

$$\begin{pmatrix} d_1 & a_1 & 0 & 0 & \cdots & 0 \\ b_2 & d_2 & a_2 & 0 & \cdots & 0 \\ 0 & b_3 & d_3 & a_3 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & b_{n-1} & d_{n-1} & a_{n-1} \\ 0 & 0 & \cdots & 0 & b_n & d_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{n-1} \\ c_n \end{pmatrix}$$

13.2 Aşağıdaki şekilde verilen üç köşegenli matris sistemini Gauss yoketme tekniği ile (13.1)'deki soruda yazdığınız programı kullanarak çözünüz.

$$\begin{pmatrix} -2 & 1 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & -2 & 1 \\ 0 & 0 & \cdots & 0 & 1 & -2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_9 \\ x_{10} \end{pmatrix} = \begin{pmatrix} -0.5 \\ -1.5 \\ -1.5 \\ \vdots \\ -1.5 \\ +0.5 \end{pmatrix}$$

(Bazılarının cevapları: $x_1 = 6.4091$, $x_5 = 25.0456$, $x_8 = 16.2728$)

13.3 $N \times N$ boyutundaki bir **A** matrisinin simetrik olup olmadığını kontrol eden bir program yazınız.

13.4 $N \times N$ boyutundaki bir **A** matrisinin transpoz'unu veren bir program yazınız.

13.5 Bir $A(N\times N)$ matrisinin köşegeninin üstünde kalan elemanlarının ortalamasını veren bir program yazınız.

13.6 Aşağıda verilen **A** ve **B** matrislerini (i) en genel şekilde (herhangi bir N değeri için) kuracak ve (ii) matrisleri kurduktan sonra çarpımını hesaplayacak ($\mathbf{C}=\mathbf{A}\times\mathbf{B}$) ve (iii) çarpım matrisini ekrana yazdıracak bir program yazınız. NOT: Program, matris çarpma işlemini yapmak için bir `SUBROUTINE` içermelidir.

$$\mathbf{A}=\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 3 & \cdots & N \\ 1^2 & 2^2 & 3^2 & \cdots & N^2 \\ 1^3 & 2^3 & 3^3 & \cdots & N^3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1^{N-1} & 2^{N-1} & 3^{N-1} & \cdots & N^{N-1} \end{pmatrix} \quad \mathbf{B}=\begin{pmatrix} 1^1 & 0 & 0 & 0 & \cdots & 0 \\ 1^2 & 2^3 & 0 & 0 & \cdots & 0 \\ 1^4 & 2^5 & 3^6 & 0 & \cdots & 0 \\ 1^7 & 2^8 & 3^9 & 4^{10} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 2^{m+2} & 3^{m+3} & 4^{m+4} & \cdots & N^{m+N} \end{pmatrix}$$

13.7 Aşağıdaki matrisleri oluşturan program parçalarını yazınız.

$$\mathbf{A}=\begin{pmatrix} -4 & 1 & 0 & 0 & \cdots & 0 \\ 1 & -4 & 1 & 0 & \cdots & 0 \\ 0 & 2 & -4 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 2 & -4 & 1 \\ 0 & 0 & \cdots & 0 & 1 & -4 \end{pmatrix}_{n\times n} \quad \mathbf{B}=\begin{pmatrix} d_1 & a_1 & 0 & 0 & \cdots & 0 \\ b_2 & d_2 & a_2 & 0 & \cdots & 0 \\ 0 & b_3 & d_3 & a_3 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & b_{n-1} & d_{n-1} & a_{n-1} \\ 0 & 0 & \cdots & 0 & b_n & d_n \end{pmatrix}_{n\times n}$$

$a_i=b_i=i$ ve $d_i=-(2i+(1+i/n)/n)$ olarak verilmektedir.

13.8 `VERI.DAT` isimli bir kütükten sırasıyla matris boyutu `N` ve matrisin elemanları girilen (a_{ij}) bir **A** matrisi için aşağıdaki şekilde tanımlanan matrisel işlemi yapan bir Fortran programı yazılmak isteniyor. Bu işlem yapılırken $\mathbf{A}^2$ matrisinin hesabını bir `SUBROUTINE` ile hesaplayıp sonuç **D** matrisini ekrana yazdıran programı yazınız. NOT: Burada **I** birim matrisi göstermektedir.

$$\mathbf{D}=2\mathbf{I}-\mathbf{A}+\frac{1}{5}\mathbf{A}^2$$

13.9 `VERI.DAT` isimli bir kütükten sırasıyla N boyutlu bir **A** matrisinin önce boyutunu sonra elemanlarını okuyup, her satırını, o satırın mutlak değerce en büyük elemanının mutlak değerini alıp, bu değere bölerek oluşacak yeni matrisi veren bir program yazınız.

13.10 $n\times n$ boyutlu **A** ve **B** matrislerini `MATRIS.DAT` isimli kütükten okuyan ve aşağıdaki işlemleri yapan bir program yazınız. Modüler programlamaya mümkün olduğunca özen gösteriniz.

$$\mathbf{A}^{-1}\times\mathbf{B}\times\mathbf{A}\times\mathbf{B}^{-1}$$

13.11 Yukarıdaki problemi, işlemleri tek indisli değişken kullanarak hesaplayan bir program

yazınız.

13.12 Bir A matrisinin elemanları için aşağıdaki ifadeyi hesaplayan bir program yazınız. Matrisin kare ve boyutunun n olduğunu varsayınız.

$$\sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{i,j}}$$

13.13 (13.6) ile verilen soruda verilen **A** ve **B** matrislerini esas alarak ve aşağıda verilen

$$x = \left(1, 1/2, 1/3, \ldots, 1/N\right)^T$$

x vektörü kullanarak, aşağıdaki işlemleri yapan bir program yazınız.

$$\left(\mathbf{A}-\mathbf{B}\right)x - \mathbf{I}x$$

NOT: **I** matrisinin birim matris olduğuna dikkat ediniz.

13.14 Bir **A** matrisinin elemanlarının 0 ila 2 arasında değiştiği biliniyor. Matrisi bilgisayara `VERI.DAT` isimli kütükten okuyan, elemanları okudukça değerinin belirtilen aralıkta olup olmadığını kontrol eden; daha sonra elemanlarının ortalamasını ve standart sapmasını hesaplayan bir program yazınız.

13.15 Bir matrisin her satırdaki elemanlarının mutlak değerce toplamını hesaplayıp, bu değerleri bir vektörün o satırdaki değerine atayan bir program yazınız.

13.16 Aşağıda şekilde verilen **A** matrisini kuran ve bu matris için $\mathbf{A}^2$ ve $\mathbf{A}^3$ matrislerini hesaplayan ve **A** matrisi ile beraber ekrana yazan bir program yazınız. *Not:* Matris, atama deyimleri şeklinde girilmeyecektir. Bir matrisin çıktısını veren bir `YAZDIR` isimli `SUBROUTINE` hazırlayınız ve bunu kullanınız

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ -2 & 1 & 2 & 3 & 4 \\ -3 & -2 & 1 & 2 & 3 \\ -4 & -3 & -2 & 1 & 2 \\ -5 & -4 & -3 & -2 & -1 \end{pmatrix}$$

13.17 Aşağıdaki şekilde verilen **A** ve **B** matrisleri için

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 5 \\ 1 & 2 & 0 & 2 & 5 \\ 1 & 0 & 3 & 0 & 5 \\ 1 & 4 & 0 & 4 & 5 \\ 1 & 0 & 0 & 0 & 5 \end{pmatrix} \quad \mathbf{B} = \begin{pmatrix} 1 & 0 & 3 & 0 & 5 \\ 0 & 1 & 0 & 3 & 0 \\ -3 & 0 & 1 & 0 & 3 \\ 0 & -3 & 0 & 1 & 0 \\ -5 & 0 & -3 & 0 & 1 \end{pmatrix}$$

aşağıda verilen matris denklemini hesaplayan bir program yazınız

$$\mathbf{A}^{-1} \times \left(\mathbf{B}-\mathbf{A}\right) \times \left(\mathbf{A}+\mathbf{B}\right) \times \mathbf{B}^{-1}$$

BÖLÜM 14
İNTEGRAL HESAP

14.1 YAMUKLAR KURALI

$f(x)$ fonksiyonu $a \le x \le b$ aralığında integre edilebilir bir fonksiyon olsun ve $f(x)$ fonksiyonunun belirtilen aralıkta belirli integralinin hesaplanmasını arzu edelim.

$$I = \int_a^b f(x)dx$$

Bu $[a,b]$ aralığını genişliği h olan N eşit parçaya bölelim. Bu durumda aralık genişliği, $h = (b-a)/N$ ifadesinden hesaplanır.

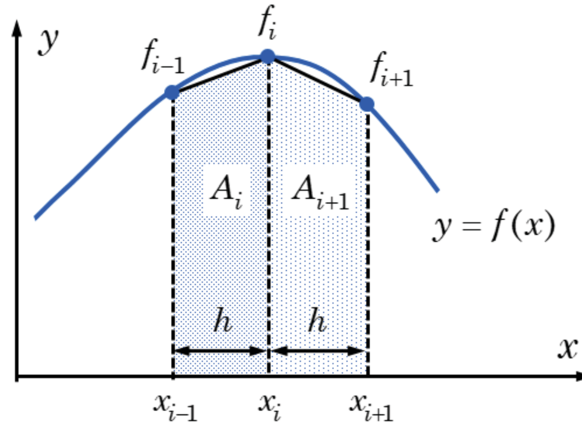
Bu fonksiyon ve oluşturulan alt aralıklar Şekil 14. 1'de verilmektedir. Şimdi Şekil 14.1'deki gibi sadece iki alt aralığı göz önüne alalım. Burada f_{i-1}, f_i, f_{i+1} noktaları birer doğru ile birleştirildiklerinde bu doğrular ile $f(x)$ fonksiyonuna bir yaklaşımda bulunulmuş olup, doğrular $f(x)$ için x_{i-1} ile x_i ve x_i ile x_{i+1} aralıklarında basit interpolasyon polinomları olarak hizmet ederler. Doğruların altında kalan alanları (yamukların alanlarını) hesaplarsak,

$$A_i = \int_{x_{i-1}}^{x_i} f(x)dx \approx h\left(\frac{f_{i-1} + f_i}{2}\right)$$

ve

$$A_{i+1} = \int_{x_i}^{x_{i+1}} f(x)dx \approx h\left(\frac{f_i + f_{i+1}}{2}\right)$$

yazılabilir.



Şekil 14.1 x_i civarında yamukların oluşturulması.

x_{i-1} ile x_{i+1} aralığında fonksiyonun integrali bu iki yamuğun alanları toplamına eşittir.

$$\int_{x_{i-1}}^{x_{i+1}} f(x)dx = A_i + A_{i+1} = \int_{x_{i-1}}^{x_i} f(x)dx + \int_{x_i}^{x_{i+1}} f(x)dx$$

Yukarıdaki denklemlerin yardımıyla integral aşağıdaki şekli alır.

$$\int_{x_{i-1}}^{x_{i+1}} f(x)dx \approx \frac{h}{2}\left(f_{i-1} + 2 f_i + f_{i+1}\right)$$

Bu son denklemi bütün alt aralıklara uygularsak, genişletilmiş Yamuklar kuralı yaklaşımı,

$$\int_a^b f(x)dx \approx \frac{h}{2}\left(f_0 + 2 f_1 + 2 f_2 + \ldots + 2 f_{N-1} + f_N \right)$$

veya seri toplamı şeklinde

$$\int_a^b f(x)dx \approx \frac{h}{2}\left(f_0 + f_N + 2\sum_{i=1}^{N-1} f_i \right)$$

olarak yazılabilir. Burada $f_0 = f(a)$ ve $f_N = f(b)$'yi ifade etmektedir.

ÖRNEK 1. Aşağıdaki integrali [0,1] aralığını 5 eşit parçaya bölerek ve Yamuklar Kuralı ile hesaplayınız. Bu integral değerini analitik integral değeri ile karşılaştırınız.

$$\int_0^1 x\left(x^2+3\right)^3 dx$$

Bu integrali hesaplamak için ilk önce aralık genişliğini saptayalım, aralık genişliği $h = (b-a)/N$ ifadesinden $h = 1/5 = 0.2$ bulunur. Aralığın beş eşit parçaya bölünmesi sonunda altı nokta ve bu noktalara ait fonksiyon değerlerinin bilinmesi gereksinimi ortaya çıkar.

Fonksiyonun değerlerini aralık içinde 0.2'lik artırımlar için hesaplayalım. Daha sonra, x_i'ler ve f_i'leri hesaplayalım. Hesaplanan bu değerleri Tablo l.'deki gibi bir tabloda toparlayalım.

Tablo l 'de fonksiyon değerlerinin yanısıra son sütundaki değerleri de verilmiştir. Dolayısıyla son sütundaki toplam integral ifadesindeki köşeli parantez içinde yer alan ifadeye eşittir. Yani

$$f_0 + f_5 + 2\sum_{i=1}^{4} f_i = 223.16672$$

Daha sonra, bu değeri integral formülünde yerine koyarsak,

$$\dot{I}ntegral = \frac{h}{2}(223.16672) = \frac{0.2}{2}(223.16672) = 22.316672$$

bulunur.

Integralin gerçek değeri 21.875 olduğuna göre yapılan mutlak hata $(|21.875 - 22.316672|)$ 0.441672 dir.

Tablo 14.1 Örnek l ile verilen problem için hazırlanan tablo.

i	x_i	f_i	işlem	Sonuç
0	0.0	0.00000	$1\times f_0 =$	0.00000
1	0.2	5.61890	$2\times f_1 =$	11.23780
2	0.4	12.62180	$2\times f_2 =$	25.24360
3	0.6	22.75986	$2\times f_3 =$	45.51972
4	0.8	38.58280	$2\times f_4 =$	77.16560
5	1.0	64.00000	$1\times f_5 =$	64.00000
			Toplam=	223.16672

Buradaki aralık sayısını (yani N değerini) artırarak daha hassas integral değeri elde edilir; çünkü aralık sayısının artmasıyla oluşturulan yamukların sayısı artırılır. Bu da eğri altında kalan alanın daha gerçekçi hesabına neden olur.

Aralık sayısı 10, 20, 50 ve 100 olarak alındığında hesaplanan integral değerleri sırasıyla 21.9857, 21.9027, 21.8794 ve 21.8761 olmaktadır. Görüldüğü üzere aralık sayısının artmasıyla integral değeri gerçek değer olan 21.875 değerine yaklaşmaktadır.

ÖRNEK 2: Yamuklar kuralı ile bir $f\left(x\right)$ fonksiyonunun $\left[a,b\right]$ aralığındaki belirli integralini hesaplayan programı.

```
MODULE bir
  CONTAINS
     REAL FUNCTION f(x)
     REAL, INTENT(IN) :: x
         f=x*(x*x+3.)**3
     END FUNCTION f
END MODULE bir

PROGRAM Yamuklar_Kurali
! ---------------------------------------------------
! YAMUKLAR KURALI İLE İNTEGRAL HESABI YAPAN PROGRAMDIR.
!   f(x)  : İNTEGRALİ HESAPLANACAK FONKSIYON
!           MODULE alt program olarak verilmiştir.
!   n     : EŞİT ARALIK SAYISI
!   h     : ARALIK GENIŞLİĞİ
!   a,b   : INTEGRAL SINIRLARI
! ---------------------------------------------------
USE bir
IMPLICIT NONE
REAL :: h, topla, integral, a=0.0, b=1.0
INTEGER :: i, n=100
!
h=(b-a)/REAL(n)            !  h=(b-a)/n
integral=0.5*(f(a)+f(b))
```

```
DO i=1,n-1
   topla=topla + f(a+i*h) ! ∑_{i=1}^{N-1} f(a+ih)
END DO
integral=(integral+topla)*h
WRITE(*,50) integral
50 FORMAT(/5X,'Yamuklar Kuralı ile integral= ',f12.7/)
END PROGRAM Yamuklar_Kurali
```

14.2 SİMPSON KURALI

Simpson kuralı ile integral işleminde, her bir alt aralık bir doğru yerine o aralık için bir parabol ile birleştirilerek, parabol altında kalan alan hesap edilir. Bu kısımda simpson formülünün türetilmesinden ziyade sonucunu yazıp bunu kullanacağız.

$$I = \frac{h}{3}\left(f_0 + f_N + 4\sum_{\substack{i=1 \\ i\ \text{tek}}}^{N-1} f_i + 2\sum_{\substack{i=2 \\ i\ \text{çift}}}^{N-2} f_i \right) \tag{14.1}$$

Denklem (14.1) ile verilen bu formül $[a,b]$ aralığını kapsar. Bu denklemin programlandığı bir program, Örnek l için hazırlanmıştır. Bu program, aritmetik deyim fonksiyonu değiştirilerek başka fonksiyonların integralinin hesaplanmasında, pekala, kullanılabilir.

ÖRNEK 3: Herhangi bir *f(x)* fonksiyonunun integralini [*a, b*] aralığında belirli integralini hesaplayan FORTRAN Programı.

Örnek 2'de verilen `MODULE bir` bu programda da aynen kullanılmaktadır.

```
PROGRAM Simpson_Kurali
! -----------------------------------------------------
! SİMPSON KURALI İLE İNTEGRAL HESABI YAPAN PROGRAMDIR.
!   f(x)  : İNTEGRALİ HESAPLANACAK FONKSİYON
!           MODULE alt program olarak verilmiştir.
!   n     : EŞİT ARALIK SAYISI
!   h     : ARALIK GENİŞLİĞİ
!   a,b   : INTEGRAL SINIRLARI
! ------------------------------------------------------
USE bir
IMPLICIT NONE
REAL :: h, x, toplam1, toplam2, uclar, integral
REAL :: a=0.0, b=1.0
INTEGER :: i, n=51
!
h=(b-a)/REAL(n)    ! h=(b-a)/n
toplam1= 0.0
toplam2= 0.0
uclar  = f(a)+f(b)
```

```
!   Tek indislilerin toplamı
DO i=1,n-1,2
   x=a+i*h
   toplam1=toplam1 + f(x)
END DO
!   Çift indislilerin toplamı
DO i=2,n-2,2
   x=a+i*h
   toplam2=toplam2 + f(x)
END DO
integral=h*(uclar+4.0*toplam1+2.0*toplam2)/3.0
WRITE(*,50) integral
50 FORMAT(/5X,'Simpson Kuralı ile integral = ',f12.7/)
END PROGRAM Simpson_Kurali
```

Bu program çalıştırıldığında N=6 için elde edilen değerin 21.8782 olduğu görülecektir. Bu değer yamuklar kuralı ile ve N=50 için hesaplanan değerden daha iyi bir sonuçtur. N=10 ve 20 için elde edilen integral değerleri sırasıyla 21.87541 ve 21.87505 olmaktadır. N=20 için neredeyse analitik olarak hesaplanan değer hesaplanmaktadır.

Sonuç olarak integral hesabında Simpson metodunun Yamuklar metodundan daha hassas olduğunu söyleyebiliriz.

ÖRNEK 4: Aşağıda verilen iki katlı integrali Simpson kuralı ile integralini hesaplayan bir program yazınız.

$$\int_{x=0}^{1}\int_{y=-1}^{1} \sin(x^2+y^2)dydx$$

Sözkonusu integrali hesaplayan bir program yazmadan önce bu sayısal işlemi nasıl yapacağımızı belirleyelim:

$$\int_{x=0}^{1}\left\{\int_{y=-1}^{1} \sin(x^2+y^2)dy\right\}dx = \int_{x=0}^{1} g(x)dx$$

Yukarıda parantez içinde yer alan *y* üzerinden integral hesaplandığında geriye sadece x üzerinden integral kalacaktır. Bu nedenle, iki kez Simpson kuralını aşağıdaki şekilde uygulayacağız.

$$\int_{x=0}^{1} g(x)dx \cong \frac{\Delta x}{3}\left[g_0 + g_{nx} + 4\sum_{\substack{i=1\\tek}}^{nx-1} g_i + 2\sum_{\substack{i=2\\çift}}^{nx-2} g_i\right]$$

burada

$$g_i = \int_{y=-1}^{1} f(x_i,y)dy \cong \frac{\Delta y}{3}\left[f_{i,0} + f_{i,ny} + 4\sum_{\substack{j=1\\tek}}^{ny-1} f_{i,j} + 2\sum_{\substack{j=2\\çift}}^{ny-2} f_{i,j}\right]$$

ve

$$f_{i,j} = f(x_i,y_j) = \sin(x_i^2 + y_j^2)$$

olarak alınmıştır.

```
PROGRAM Ornek4
IMPLICIT NONE
INTEGER, PARAMETER :: nx=20, ny=40
INTEGER :: i, j
REAL, DIMENSION(0:nx) :: x, fx
REAL, DIMENSION(0:ny) :: y, fy
REAL :: a=0.0, b=1.0, c=-1., d=1.
REAL :: dx, dy, integral, FXY
!
dx=(b-a)/REAL(nx)  ! Δx = (b − a)/nx
dy=(d-c)/REAL(ny)  ! Δy = (d − c)/ny
DO i=0,nx
   x(i)=a+REAL(i)*dx       ! x_i = a + iΔx
   DO j=0,ny
      y (j)=c+REAL(j)*dy  ! y_j = c + jΔy
      fy(j)=FXY(x(i),y(j))! g_j = f(x_i, y_j)
   END DO
   CALL Simpson_Kurali(ny,dy,fy,fx(i)) ! g_i'ler
END DO
CALL Simpson_Kurali(nx,dx,fx,integral) ! ∫_0^1 g(x)dx
! Sonucu yazdır
WRITE(*,*) integral
END PROGRAM Ornek4

REAL FUNCTION FXY(x,y)
IMPLICIT NONE
REAL, INTENT(IN) :: x, y
  FXY=SIN(x*x+y*y)
END FUNCTION FXY

SUBROUTINE Simpson_Kurali(n,h,fonk,integral)
IMPLICIT NONE
!  Simpson kuralına göre integrali hesaplayan
!  Alt Programdır.
!  ----------------------
!  n    : Aralık sayısı
!  h    : Aralık genişliği
!  fonk : Fonksiyon değerleri, i=0,n
!  integral : İntegral değeri
!
REAL, DIMENSION(0:n), INTENT(IN) :: fonk
REAL, INTENT(IN) :: h
INTEGER, INTENT(IN) :: n
REAL, INTENT(OUT) :: integral
REAL :: toplam1, toplam2, uclar
INTEGER :: i
!
uclar=fonk(0)+fonk(n)
```

```
!   Tek indislilerin toplamı
toplam1=0.0
DO i=1,n-1,2
   toplam1=toplam1+fonk(i)
END DO
!   Çift indislilerin toplamı
toplam2=0.0
DO i=2,n-2,2
   toplam2=toplam2+fonk(i)
END DO
integral=h*(uclar+4.0*toplam1+2.0*toplam2)/3.0
END SUBROUTINE Simpson_Kurali
```

Programını çeşitli aralık sayıları için alıştırdığımızda aldığımız sonuçlar aşağıdaki Tablo 14.2'de özetlenmiştir. Her integral değişkeni için aralık sayısı artırıldığında, integralin önce ilk üç sonra ilk dört ve en sonunda `nx=50` ve `ny=100` için ilk beş basamağının sabitlendiği, yani yakınsadığı, görülmektedir.

Tablo 14.2 Örnek 4'de verilen problemin farklı aralık sayıları için sonuçları.

nx	ny	integral
5	5	0.59582
10	10	1.12232
10	20	1.12255
20	20	1.12256
50	100	1.12258

14.3 GAUSS-LEGENDRE KUADRATÜRLERİ VE İNTEGRASYONU

Gauss-Legendre kuadratürleri Legendre polinomlarının köklerininin absis ve ağırlıklarının hesabına dayanan bir kuadratür yöntemidir. Legendre polinomlarının genel formu

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2-1)^n, \quad n \geq 0$$

ile verilir. $P_0(x)=1$, $P_1(x)=x$, $P_2(x)=(3x^2-1)/2$,.. şeklinde türetilebilir.

N-nokta Gauss-Legendre integrasyonu

$$\int_{-1}^{1} f(x)dx \cong \sum_{i=1}^{N} w_i f(x_i)$$

olarak tanımlanır. Burada x_i (absisler) ve w_i (ağırlıklar) çiftine Gauss-Legendre kuadratürleri adı verilir. x_i'ler, $P_N(x)=0$ polinomunun kökleridir. w_i ağırlıkları da

$$w_i = \frac{2(1-x_i^2)}{\left[NP_{N-1}(x_i)\right]^2}$$

bağıntısından hesaplanır. Ancak absis ve ağırlıklar (-1,1) aralığında integral alma işlemine uygundur. Bu kuadratürleri (*a*,*b*) aralığında integral alma işlemi için uyarladığımızda

$$x_i \leftarrow \frac{b-a}{2} x_i + \frac{b+a}{2}, \qquad w_i \leftarrow \frac{b-a}{2} w_i$$

olarak değiştirilmelidir.

Gauss-Legendre kuadratürlerin (-1,1) aralığı için hesaplayan bir algoritma aşağıda verilmiştir:

```
SUBROUTINE Gauss_KURALI(n,x,w)
IMPLICIT NONE
!
!   (-1,1) aralığı için n-Nokta Gauss-Legendre
!   Kuadratürlerini hesaplayan alt programdır
!
INTEGER, INTENT(IN) :: n                ! Nokta sayısı
REAL, DIMENSION(N), INTENT(OUT) :: x   ! Kuadratür absisleri
REAL, DIMENSION(N), INTENT(OUT) :: w   ! Kuadratür ağırlıkları
REAL    :: pi, e1, x0, t, pk, pk1, pkm1, t1, d1, &
       den, dpn, u, v, h, p, dp, fx, d2pn, d3pn, &
       d4pn, pkp1
INTEGER :: i, m, k, mi, ii, mf
pi = 4.0 * ATAN(1.0)
m  = (n+1)/2
e1 = REAL(n*(n+1))
DO i=1,m
   t   = (4.*i-1.)*pi/(4.*n+2.)
   x0  = (1.0-(1.0- 1.0/n)/(8.0*n*n))*COS(t)
   pkm1= 1.0
   pk  = x0
   DO k=2,n
      t1   = x0 * pk
      pkp1 = t1 - pkm1 -(t1-pkm1)/REAL(k)+t1
      pkm1 = pk
      pk   = pkp1
   END DO
   den  = 1.0 - x0 * x0
   d1   = n * ( pkm1 - x0 * pk )
   dpn  = d1 / den
   d2pn = (2.0*x0*dpn-e1*pk)/den
   d3pn = (4.0*x0*d2pn+(2.0 -e1)*dpn)/den
   d4pn = (6.0*x0*d3pn+(6.0 -e1)*d2pn)/den
   u = pk / dpn
   v = d2pn / dpn
   h = -u*(1.0+.50*u*(v+u*(v*v-u*d3pn/(3.0*dpn))))
   p = pk+h*(dpn+.50*h*(d2pn+h/3.0*(d3pn+.25*h*d4pn)))
   dp= dpn + h * (d2pn+.50*h*(d3pn+h*d4pn/3.0))
   h = h - p/dp
   x(i) = x0 + h
   fx=d1-h*e1*(pk+.5*h*(dpn+h/3.*(d2pn+.25*h*(d3pn+&
     0.2*h*d4pn))))
   w(i) = 2.0*(1.0-x(i)*x(i))/(fx*fx)
END DO
```

```
IF(m+m>n) THEN
    x(m) = 0.0
    mf = m - 1
ELSE IF(m+m==n) THEN
   mf = m
END IF
DO i=1,mf
   x(i) = -x(i)
END DO
DO i=1,mf
   mi = m + i
   ii = m - i + 1
   IF(mf==m-1) ii = m-i
   x(mi) =-x(ii)
   w(mi) = w(ii)
END DO
END SUBROUTINE Gauss_KURALI
```

14.4 İNTEGRAL YÖNTEMLERİNİN KUADRATÜR FORMUNA DÖNÜŞTÜRÜLMESİ

Bütün integral yöntemlerini aşağıdaki kuadratür formunda yazmamız mümkündür:

$$\int_{x=a}^{b} f(x)dx \cong \sum_{i=1}^{N} w_i f(x_i)$$

Burada yönteme uygun kuadratürlerin tesbiti önemli olmaktadır.

N-nokta yamuklar kuralında ($N-1$ aralık)

$$\int_{x=a}^{b} f(x)dx \cong \frac{h}{2} f(a) + hf(a+h) + hf(a+2h) + \cdots + hf(a+(N-1)h) + \frac{h}{2} f(b)$$

olarak belirtilebildiğinden genel kuadratür formuyla kıyaslama yaparak

$$h = \frac{b-a}{N-1}, \quad x_i = a+(i-1)h, \quad w_i = \begin{cases} h & 2 \le i \le N-1 \\ h/2 & i=1 \text{ ve } i=N \end{cases}$$

olduğu görülür. Aynı şekilde *N*-nokta Simpson kuralı için

$$h = \frac{b-a}{N-1}, \quad x_i = a+(i-1)h, \quad w_i = \begin{cases} h/3 & i=1 \text{ ve } i=N \\ 4h/3 & i=2,4,6,\ldots \\ 2h/3 & i=3,5,7,\ldots \end{cases}$$

olarak yazılabilir.

ÖRNEK 5: Yamuklar, Simpson ve Gauss-Legendre yöntemlerine göre herhangi bir (a,b) aralığında N-integrasyon noktası kullanarak kuadratür absislerini ve ağırlıklarını getiren bir program yazınız.

Yukarıda bahsedilen kuadratürleştirme formülasyonlarının programlanmasından ibarettir:

```
SUBROUTINE KUAD(tip, n, a, b, x, w)
!  (a,b) aralığında integral almak için çeşitli integral
!  yöntemlerine göre kuadratürleri hesaplayan alt
!  programdır
!     Tip=0 Yamuklar, Tip=1 Simpson, Tip=2 Gauss-Legendre
IMPLICIT NONE
INTEGER, INTENT(IN) :: Tip  ! Yöntem tipi
INTEGER, INTENT(IN) :: n    ! Nokta sayısı
REAL, INTENT(IN)    :: a, b ! İntegrasyon aralığı
REAL, DIMENSION(N), INTENT(OUT) :: x ! Absisler
REAL, DIMENSION(N), INTENT(OUT) :: w ! Ağırlıklar
INTEGER :: i                ! İndis değişkeni
REAL    :: h                ! Absilerin aralık genişliği
SELECT CASE (Tip)
    CASE (:-1)
      PRINT*, ' Nokta sayısı NEGATİF olamaz!'
    CASE (0)  !  Yamuklar kuralı
      h=(b-a)/REAL(n-1)
      x(1)=a
      DO i=2,n
         x(i)=x(i-1)+h
         w(i)=h
      END DO
      w(n)=w(n)*0.5
      w(1)=w(n)
    CASE (1)  !  Simpson kuralı
      h=(b-a)/REAL(n-1)
      DO i=1,n
         x(i)=a+(i-1)*h
      END DO
      DO i=2,n-1,2
         w(i)=4.0*h/3.0
      END DO
      DO i=3,n-2,2
         w(i)=2.0*h/3.0
      END DO
      w(1)=h/3.0
      w(n)=h/3.0
    CASE (2)  !  Gauss-Legendre kuadratürleri
      CALL Gauss_KURALI(n,x,w)
      DO i=1,n
         x(i)=0.5*(b-a)*x(i)+0.5*(b+a)
         w(i)=0.5*(b-a)*w(i)
      END DO
    CASE DEFAULT
      PRINT*, 'Yöntem tanımlanmamış...!'
END SELECT
END SUBROUTINE KUAD
```

ÖRNEK 6: Aşağıda verilen fonksiyonun nümerik integralini Yamuklar, Simpson ve Gauss-Legendre yöntemlerini kullanarak hesaplayan bir program yazınız. Yamuklar ev Simpson yöntemlerinde 21 nokta, Gauss-Legendrede ise 5 nokta kullanınız.

$$\int_0^1 \frac{e^{x^2}}{1+x^2}\,dx$$

Öncelikle integrali alıncak fonksiyonu `FUNCTION` olarak tanımlamak gerekir. Daha sonra Örnek 4'de verilen `KUAD` programında `Tip` için değerler değiştirilerek integral hesaplanabilir. Bu programda `Gauss_KURALI` isimli alt programdan da yararlanılmıştır.

```
PROGRAM kuadraturler
IMPLICIT NONE
REAL, DIMENSION(21) :: x, w
INTEGER :: i, n=21
REAL :: a=0.0, b=1.0, topl, f
CALL KUAD(0,n,a,b,x,w) ! Tip=0 Yamuklar Kuralı
topl=0.0
DO i=1,n
  topl=topl+w(i)*f(x(i))
END DO
PRINT*, 'integral (yamuklar kuralı)=',topl
CALL KUAD(1,n,a,b,x,w) ! Tip=1 Simpson Kuralı
topl=0.0
DO i=1,n
  topl=topl+w(i)*f(x(i))
END DO
PRINT*, 'integral (Simpson kuralı) =',topl
n=5
CALL KUAD(2,n,a,b,x,w) ! Tip=2 Gauss-Legendre Kuad
topl=0.0
DO i=1,n
  topl=topl+w(i)*f(x(i))
END DO
PRINT*, 'integral (Gauss-Legendre) =',topl
END PROGRAM kuadraturler

REAL FUNCTION f(x)
REAL, INTENT(IN)::x
     f=EXP(x*x)/(1+x*x)
END FUNCTION f
```

Programının çıktısı

```
integral (yamuklar kuralı)= 1.07623792
integral (Simpson kuralı) = 1.07595527
integral (Gauss-Legendre) = 1.07595479
```

olarak elde edilir.

Her üç yöntemle farklı nokta sayıları kullanılarak hesaplanan integral değerleri Tablo 14.3'de verilmiştir. Bu tablodaki değerler incelendiğinde nokta sayısı artırıldığında integral değeri, gerçek değere yakınlaştığını, basamak doğruluğunun arttığını gözleriz. En az nokta sayısı kullanarak gerçek integral değerine en yakın sonuç veren yöntem Gauss-Legendre kuadratür yöntemidir. Üç-nokta Gauss-Legendre kuadratürleri ile integral 3 basamak doğrulukla elde edilmiştir ki elle hesaplanacak kadar basitlik sağlar. Yamuklar ve Simpson yöntemleri arasında bir kıyaslama yaptığımızda Simpson yönteminin Yamuklar kuralına göre daha iyi sonuçlar ürettiğini görmekteyiz. 21 nokta kullanıldığında Yamuklar iki basamak doğruluk sağlarken Simpson kuralı 5 basamak doğruluğa ulaşmıştır. Bunun nedenlerine Nümerik Analiz dersinde değinilecektir.

Tablo 14.3 Değişik nokta sayıları için yöntemlerin performansı.

N	Yamuklar	Simpson	*N*	Gauss-Legendre
6	1.080461	1.0713601	2	1.0746690
11	1.077086	1.0759608	3	1.0758653
16	1.076458	1.0751362	4	1.0759559
21	1.076238	1.0759553	5	1.0759548

ALIŞTIRMALAR

14.1 Aşağıdaki şekilde kesikli dağılımı verilen fonksiyonun integralini yamuklar kuralı ve Simpson kurallarına göre hesaplayınız.

i	0	1	2	3	4	5	6	7	8
x_i	1.0	1.25	1.50	1.75	2.0	2.25	2.50	2.75	3.0
f_i	1.0	1.58	3.74	4.88	8.57	8.32	7.36	7.96	8.84

14.2 Aşağıdaki şekilde kesikli dağılımı verilen fonksiyonun integralini, yamuklar kuralı ve Simpson kurallarına göre integralini hesaplayan program yazınız.

i	0	1	2	3	4	5	6
x_i	0.50	0.65	0.80	0.95	1.10	1.25	1.40
f_i	1.6487	1.9155	2.2255	2.5857	3.0041	3.4903	4.0552

$$\int_{0.5}^{1.40} \left(1+x^2\right) f\left(x\right) dx$$

14.3 Aşağıdaki integralleri hesaplayan programlar yazınız.

$$\text{(a)} \int_0^{0.8} e^{-x^2} dx \qquad \text{(b)} \int_0^{\pi} \ln\left(5-4\cos x\right) dx$$

14.4 Aşağıdaki şekilde verilen fonksiyonu herhangi bir x ve n değeri için hesaplamak istiyoruz. Bu integrali aralığı 20 eşit parçaya bölerek ve yamuklar kuralını kullanarak hesaplayan bir program yazınız.

$$W_n = \frac{1}{\pi} \int_0^{\pi} \sin\left(n\theta - x\sin\theta\right) d\theta$$

14.5 Aşağıdaki integrali, integral sınır değerlerini sonlu olacak şekilde uygun bir dönüşüm yaptıktan sonra değişik aralık sayısı için yamuklar kuralını kullanarak hesaplayan bir program yazınız.

$$\int_1^\infty e^{-1/x^2}\,\frac{dx}{x^2}$$

14.6 Herhangi bir *x* değeri için *Z* yi hesaplayan bir program yazınız. *U* ile verilen serinin hesabında, serinin hata mertebesi 10^{-4}'den küçük olsun. Ayrıca *V* ile gösterilen integrali de Simpson kuralına göre aralığı 50 parçaya bölerek hesaplayınız.

$$U=\frac{x}{2}+\frac{x^2}{3}+\frac{x^3}{4}+\frac{x^5}{5}+\ldots+\frac{x^m}{m+1}+\ldots=\sum_{n=1}^{\infty}\frac{x^n}{n+1}$$

$$V=\int_{0.1}^{x}\frac{\sin t}{t}dt \qquad Z=\frac{U+V}{UV}$$

14.7 Aşağıdaki şekilde verilen fonksiyonu belirtilen limitler için hesaplamak istiyoruz. Bu integrali, aralığı 100 eşit parçaya bölerek, Simpson kuralı ile hesaplayan bir program yazınız.

$$Y=\int_1^5\frac{dx}{x^x}$$

14.8 Matematikte sıkça kullanılan bir fonksiyon da hata fonksiyonudur ve aşağıdaki integral ile tanımlıdır.

$$erf\left(x\right)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-T^2}dT$$

Herhangi bir *x* değeri için hata fonksiyonunun; yani *erf*(*x*)'in, değerini hesaplayan bir program yazmak için (a) algoritma hazırlayınız, (b) akış şeması çiziniz ve (c) Fortran programı yazınız, integral yöntemi Yamuklar ve Simpson kurallarından birini seçebileceğimiz şekilde hazırlansın ve her bir yönteme göre integral hesaplayan `SUBROUTINE`'ler oluşturulsun.

14.9 Aşağıdaki integrali yamuklar kuralına göre herhangi bir *M* için hesaplayan bir program yazınız.

$$\int_{\pi/8}^{\pi/4}\frac{\sec x dx}{1+\tan x}$$

14.10 Bir elektrik devresinde belirli bir şekilde dalga şekli üretmek arzu edilir. Bir dalga şekli $f\left(t\right)$ aşağıdaki şekilde verilmiş olsun.

$$f\left(t\right)=\begin{cases}50\cos\left(\pi/50\right)t, & -50\le t\le 50\\ 0, & -100\le t\le -50,\quad 50\le t\le 100\end{cases}$$

Dalga periyodu *T*'nin 200 olduğu aşikardır. Dalga şekli Fourier serisinin ilk 10 terimi ile ifade edilmek istenmektedir.

$$f\left(t\right)=a_0+2\sum_{k=1}^{9}a_k\cos\left(k\omega t\right)$$

burada $\omega=2\pi/T$ ve a_k katsayıları da

$$a_k = \frac{2}{T} \int_0^{T/2} f(t) \cos(k\omega t)\, dt; \qquad k = 0,1,2,3,\ldots$$

olarak verilmektedir. Yazacağınız bir altprogram $a_0, a_1, \ldots, a_9$ katsayılarım Simpson kuralı ile aralığı 100 eşit parçaya bölerek hesaplasın.

14.11 Yukarıdaki problemde herhangi bir *t* anında $f(t)$ dalga şeklinin değerini Fourier serisi ile hesaplayan ve analitik olarak verilen değerlerini ve aralarındaki farkı hesaplayıp, ekrana yazdıran bir program yazınız.

BÖLÜM 15
KOMPLEKS ANALİZ

15. 1 GENEL BİLGİLER

FORTRAN programlama dilini diğer dillerden ayıran en önemli özelliklerinden biri de, Fortran derleyicilerinde kompleks sayılarla yapılan tüm işlemlerin tanımlanmış olmasıdır. Kompleks sayılar $z = a + ib$ şeklinde belirlenen bir gerçek (a) ve bir de sanal (b) kısımdan oluşan bir sayı çiftidir; burada $i = \sqrt{-1}$ 'i temsil etmektedir. İki kompleks veya karmaşık sayı $z_1 = a_1 + ib_1$ ve $z_2 = a_2 + ib_2$ ele alındığında,

Toplama ve Çıkarma işlemi, $z = z_1 \pm z_2 = (a_1 \pm a_2) + i(b_1 \pm b_2)$

Çarpma işlemi, $z = z_1 \cdot z_2 = (a_1 \cdot a_2 - b_1 \cdot b_2) + i(a_1 \cdot b_2 + b_1 \cdot a_2)$

Bölme işlemi, $z = \dfrac{z_1}{z_2} = \dfrac{a_1 \cdot a_2 + b_1 \cdot b_2}{a_2^2 + b_2^2} + i\dfrac{b_1 \cdot a_2 - a_1 \cdot b_2}{a_2^2 + b_2^2}$

z'in eşleniği, $\overline{z} = a - ib$

olarak tanımlanmaktadır. Ayrıca bir kompleks sayıyı kutupsal şekilde tanımlamak mümkündür. Yani

$$z = re^{i\theta} = r(\cos\theta + i\sin\theta)$$

şeklinde de belirtilebilir. Burada θ kompleks sayının *x*-ekseni ile saat yönünün tersi yönde yaptığı açıdır ve $\theta = \arg(z)$ olarak da yazılır ve $\theta = \arctan(b/a)$ ile hesaplanır; yalnız (a,b) çiftinin *x-y* kartezyen sistemde hangi bölgeye denk geldiğini bilmek önemlidir; diğer taraftan, r ise büyüklük olup $r = |z| = \sqrt{a^2 + b^2}$ şeklinde verilmektedir.

Fortran programlama dilinde bu işlemler otomatik olarak yapılmaktadır. Örneğin iki kompleks sayı arasında toplama, çıkarma, çarpma ve bölme işlemi yapan ve ekrana yazdıran program verilmektedir. Bu program $z_1 = 1 - 3i$ ve $z_2 = 4 + 3i$ için aşağıda verilmiştir.

```
PROGRAM sanal_sayilar
IMPLICIT NONE
COMPLEX :: z1=(1.,-3.), z2=(4.,3.) ! Tip tanımlama
PRINT*, 'Z1=',z1
PRINT*, 'Z2=',z2
PRINT*, 'Z1+Z2=',z1+z2
PRINT*, 'Z1-Z2=',z1-z2
PRINT*, 'Z1*Z2=',z1*z2
PRINT*, 'Z1/Z2=',z1/z2
END PROGRAM Sanal_sayilar
```

Girdi programa (l. ,-3.) , (4. ,3.) olarak temin edilmiştir. Çıktı

```
Zl=  (1.0000000,-3.0000000)
Z2=  (4.0000000,3.0000000)
Z1+Z2= (5.0000000,0.000000)
Z1-Z2= (-3.0000000,-6.0000000)
Z1*Z2= (13.0000000,-9.0000000)
Z1/Z2= (-0.2000000,-0.6000000)
```

olarak elde edilir. Parantez içindeki ilk değer gerçek ikinci değerde sanal kısmı ifade etmektedir. Bu programı formatlı çıktı almak için aşağıdaki şekilde değiştirmek de mümkündür. Bu durumda

```
PROGRAM Sanal_sayilar
IMPLICIT NONE
COMPLEX :: z1=(1.,-3.), z2=(4.,3.)
WRITE(*,10) z1,z2,z1+z2,z1-z2,z1*z2,z1/z2
10 FORMAT(2X,'Z1=',f8.3,'+',f8.3,' i',/, &
  2X,'Z2=',f8.3,'+',f8.3,' i',/,     &
  2X,'Z1+Z2=',f8.3,'+',f8.3,' i',/,  &
  2X,'Z1-Z2=',f8.3,'+',f8.3,' i',/,  &
  2X,'Z1*Z2=',f8.3,'+',f8.3,' i',/,  &
  2X,'Z1/Z2=',f8.3,'+',f8.3,' i')
END PROGRAM Sanal_sayilar
```

Bu durumda çıktı aşağıdaki gibi elde edilir.

```
Z1=  1.000+  -3.000 i
Z2=  4.000+   3.000 i
Z1+Z2=  5.000+   0.000 i
Z1-Z2= -3.000+  -6.000 i
Z1*Z2= 13.000+  -9.000 i
Z1/Z2= -0.200+  -0.600 i
```

Tablo 15.1 Karmaşık sayılarla kullanılan arşiv fonksiyonları ve işlevleri (Aşağıda c = a + **i** b olarak alınmıştır).

Jenerik Fonksiyon	Tip Belirli Fonksiyon	Değeri	Bilgi
ABS(c)	**CABS**(c)	$\sqrt{a^2+b^2}$	Karmaşık sayının büyüklüğünü hesaplar; sonuç gerçek sayıdır
	AIMAG(c)	b	
CMPLX(a,b, kind)	$(a,b)\rightarrow$	$a+ib$	a ve b birleştirerek $a+ib$ karmaşık sayısını oluşturur. Kind opsiyonel bir tam sayıdır.
	CONJG(c)	$a-ib$	Karmaşık sayının eşleniğini verir
DBLE(c)			c karmaşık sayısının gerçek kısmını çift hassasiyetli bir sayıya dönüştürür.
INT(c)			c karmaşık sayısının gerçek kısmını bir tamsayıya dönüştürür.
REAL(c,kind)			c karmaşık sayısının gerçek kısmını bir gerçek sayıya dönüştürür. Kind opsiyonel bir tamsayıdır.

Hatırlayacağınız gibi kompleks sabitler ve değişkenler `COMPLEX` olarak tanımlanır. Bir kompleks sabit, örneğin $1+3i$,

```
(l.,3.)    (0.lel.3.0)     (1..3.E0)  (0.1E1,0.3E1)
```

değerlerinden herhangi birisi ile ifade edilebilir. Fakat (1,3) olarak kullanılamaz. Diğer taraftan, gerçek veya sanal kısım değişken ise, $z=x+3i$ gibi, bu durumda bu kompleks ifadeyi `(X,3.)` olarak tanımlayamayız. Buna rağmen Fortran arşiv fonksiyonlarından `CMPLX` fonksiyonu bu iş için tasarlanmıştır. Bu fonksiyonun kullanımı ile gerçek veya sanal kısım veya her ikisi de aynı anda değişken olarak tanımlanabilmektedir. Örneğin,

Doğru

```
PROGRAM ornek
IMPLICIT NONE
COMPLEX :: a, b, c
REAL :: x, y
READ*, x, y
a=CMPLX(x,2.)
b=CMPLX(-1.,y)
c=a*b
PRINT*, c
END PROGRAM Ornek
```

Yanlış

```
PROGRAM ornek
IMPLICIT NONE
COMPLEX :: a, b, c
REAL :: x, y
READ*, x, y
a=(x,2.)
b=(-1.,y)
c=a*b
PRINT*, c
END PROGRAM Ornek
```

`CMPLX` fonksiyonunun başka kullanım durumları aşağıda verilmektedir.

`Z=CMPLX(X+1.0,1.0-X)`	$z=(x+1)+(1-x)i$
`Z=CMPLX (X+Y, 3.0+Y)`	$z=(x+y)+(3+x)i$
`Z=CMPLX(X*Y,T)`	$z=xy+ti$

Aşağıda sıralanan atama deyimleri geçerli atama deyimleridir.

```
COMPLEX :: Z,X,Y
Z=(.5,0.45e+01)
Z=Z+X**2
Z=3.                 ! sanal kısmının "sıfır" olduğu anlamına gelir
Z=0.125*X
Z=2.*X+3.*Y-4.
Z=CMPLX(X,Y-1.5)
```

Kompleks arşiv fonksiyonları `REAL`, `AIMAG`, `CABS`, `CSQRT`, `CSIN`, `CCOS` gibi fonksiyonları içermektedir.

Tablo 15.1'de verilen `REAL(c)`, `c` kompleks sayısının gerçek kısmını, `AIMAG(c)` `c` kompleks sayısının sanal kısmım, `CABS(c)` ise mutlak değerini ve `CSQRT(c)`'de `c`'nin karekökünü verir.

15.2 KOMPLEKS SAYILARLA UYGULAMALAR

Mühendislik problemlerinin çözümünde özellikle fizik, elektrik, bilgisayar, makina ve kontrol mühendisliğinde kompleks sayılar ve uygulamaları önemli yer tutar. Uygulamalar bir polinomun kompleks köklerinin bulunmasından, kompleks matrisler ile işlemler (toplama, çarpma, matrisin tersi v.s), seri toplam, diferansiyel denklemin çözümü v.b şekillerinde olabilmektedir. Bu nedenle bu problemlerin Fortran90'da programlanması büyük bir kolaylık sağlamaktadır.

ÖRNEK 1: İkinci dereceden denklemin, `COMPLEX` tanımlaması ve uygulaması kullanarak, gerçek ve/veya sanal köklerini veren bir program yazınız.

Diskriminantı `COMPLEX` olarak tanımladıktan sonra ($\Delta = (b^2 - 4ac) + 0i$), sıfıra eşit, pozitif veya negatif olup olmamasını kontrol etmeye gerek kalmaz çünkü diskriminant negatif olsa da karmaşık sayılarla işlemlerde negatif sayıların kökleri vardır ve sanaldır.

```
PROGRAM Ornek1
IMPLICIT NONE
REAL :: a, b, c, delta, a2
COMPLEX :: x1,x2
WRITE(*,50)
!   Katsayıları oku
READ (*,*) a, b, c
delta = b*b-4.*a*c
a2 = 2.*a
!   Kökleri hesapla
!   √(Δ+0i)   şeklinde temsil ediyoruz
x1 = (-b+SQRT( CMPLX(delta,0.) ))/a2
x2 = (-b-SQRT( CMPLX(delta,0.) ))/a2
WRITE (*,25) a, b, c ! Katsayıları yaz
!   Kökleri gerçek ve sanal kısımlarıyla yaz
WRITE (*,75) ' x1 = ', REAL(x1), AIMAG(x1)
WRITE (*,75) ' x2 = ', REAL(x2), AIMAG(x2)
25 FORMAT(2x,F5.2,' X**2 + ',F5.2,' X + ',F5.2,' =0'/,&
  'Denkleminin Kökleri')
50 FORMAT (' A*X**2+B*X+C=0 denkleminin', &
  ' katsayılarını giriniz')
75 FORMAT (A,F8.4,' + ( ',F8.4,' ) i ')
END PROGRAM Ornek1
```

Çeşitli katsayı değerleri için çözümleri irdeleyecek olursak,

Programın $a = 1$, $b = 1$ ve $c = 1$ için çıktısı

```
  1.00 X**2 +  1.00 X +  1.00 =0
Denkleminin Kökleri
 x1 =  -0.5000 + (   0.8660 ) i
 x2 =  -0.5000 + (  -0.8660 ) i
```

Programın $a=1$, $b=-2$ ve $c=1$ için çıktısı

```
   1.00 X**2 + -2.00 X +  1.00 =0
Denkleminin Kökleri
 x1 =   1.0000 + (   0.0000 ) i
 x2 =   1.0000 + (   0.0000 ) i
```

Programın $a=1$, $b=-4$ ve $c=3$ için çıktısı

```
   1.00 X**2 + -4.00 X +  3.00 =0
Denkleminin Kökleri
 x1 =   3.0000 + (   0.0000 ) i
 x2 =   1.0000 + (   0.0000 ) i
```

elde edilir.

ÖRNEK 2: Aşağıdaki şekilde verilen bir kompleks serinin toplamını veren program yazınız. Bu serinin toplamında kıstas olarak $|z_n| < \varepsilon$ ifadesini kullanınız.

$$1+\frac{(1+i)}{1!}+\frac{(1+i)^2}{2!}+\frac{(1+i)^3}{3!}+\cdots=\sum_{n=0}^{\infty}\frac{(1+i)^n}{n!}$$

Serinin ilk iki teriminin toplamının 2 + *i* olduğu görülür. Bu durumda programda gerçek kısımların toplamını hesaplarken `toplam=1.0` alabiliriz (çünkü serinin ilk terimidir).

```
PROGRAM Ornek2
IMPLICIT NONE
INTEGER :: n=0
COMPLEX :: toplam, terim
REAL    :: f=1.0, eps=1.0E-5
!  F= faktoriyel olarak kullanılıyor
toplam=(1.0,0.0)
DO
  n = n + 1                       ! Sayaç kullanılıyor
  f = f * n                       ! n faktoriyel hesabı
  terim  = (1.,1.)**n / f
  toplam = toplam + terim
  IF(CABS(terim)<eps) EXIT        ! |z_n| < ε?
END DO
PRINT*, 'Seri Toplam=',toplam
PRINT*, 'Serinin ilk ',n,' terimi toplandı '
END PROGRAM Ornek2
```

Programının çıktısı

```
Seri Toplam= (1.4686949,2.287354)
Serinin ilk 10 terimi toplandı
```

olarak bulunur.

ÖRNEK 3: İki kompleks matrisin toplamım veren bir program yazınız ve örnek olarak aşağıdaki iki kompleks matrisi için deneyiniz.

$$\mathbf{A}=\begin{pmatrix} 1+2i & -1-i & 3i \\ -4+i & 3+2i & -1+5i \\ 3-4i & i & -1 \end{pmatrix} \qquad \mathbf{B}=\begin{pmatrix} -1-3i & 1+5i & 4+3i \\ -2+3i & 8-6i & 4i \\ -1+6i & 3+i & -3-7i \end{pmatrix}$$

Bu örnekte kompleks matrislerin çarpımı normal iki matrisin çarpımı şeklinde düşünülebilir. Programlamada karşılaşılacak tek farklılık matrislerin `COMPLEX` olarak tanımlanması olacaktır. Dolayısıyla hazırlanan programda `CMCARP` isimli altprogram Bölüm 13'teki `MCARP` ile aynı şekilde hazırlanmış; fakat `Zl`, `Z2` ve `Z3 COMPLEX` olarak tanımlanmışlardır.

```
PROGRAM Ornek3
IMPLICIT NONE
COMPLEX, DIMENSION(3,3) :: a, b, c, d
INTEGER :: i,j,n=3
OPEN(UNIT=5,FILE='girdi.dat',STATUS='old')
!    A kompleks Matrisini Kütükten Oku
READ(5,*) ((a(i,j),j=1,n),i=1,n)
!    B kompleks Matrisini Kütükten Oku
READ(5,*) ((b(i,j),j=1,n),i=1,n)
d=a+b                      ! Matrislerin toplamı D=A+B
WRITE(6,20) 'A'
   CALL Sanal_Matris_Yazdir(n,a,6)
WRITE(6,20) 'B'
   CALL Sanal_Matris_Yazdir(n,b,6)
CALL cmcarp(n,a,b,c)   ! Matris çarpımı yapıldı
WRITE(6,20) 'C'
   CALL Sanal_Matris_Yazdir(n,c,6)
WRITE(6,20) 'D'
   CALL Sanal_Matris_Yazdir(n,d,6)
20 FORMAT(/2x,A,' matrisi')
END PROGRAM Ornek3

SUBROUTINE Sanal_Matris_Yazdir(n,M,UnitNo)
! nxn boyutlu sanal matrisi yazdıran
! alt programdı
!  ** n      : matrisin boyutu
!  ** A      : nxn'lik kompleks matris
!  ** UnitNo : Yazdırma birimi numarası
IMPLICIT NONE
INTEGER, INTENT(IN) :: n, UnitNo
COMPLEX, DIMENSION(n,n), INTENT(IN) :: M
INTEGER :: i, j
DO i=1,n
   WRITE(UnitNo,10) (M(i,j),j=1,n)
END DO
10 FORMAT(3(2x,'(',F6.1,',',F6.1,')'))
END SUBROUTINE Sanal_Matris_Yazdir
```

```
SUBROUTINE cmcarp(n,a,b,c)
IMPLICIT NONE
INTEGER, INTENT(IN):: n
COMPLEX, DIMENSION(n,n), INTENT(IN)  :: a
COMPLEX, DIMENSION(n,n), INTENT(IN)  :: b
COMPLEX, DIMENSION(n,n), INTENT(OUT) :: c
INTEGER :: i,j,k
!
! C=AxB İŞLEMİNİ YAPAN ALT PROGRAM
!
! ** A : nxn BOYUTLU SANAL MATRİSTİR
! ** B : nxn BOYUTLU SANAL MATRİSTİR
! ** C : nxn BOYUTLU SANAL MATRİSTİR (=A*B)
DO i=1,n
   DO j=1,n
      c(i,j)=0.0
       DO k=1,n
          c(i,j)=c(i,j)+a(i,k)*b(k,j)
       END DO
   END DO
END DO
END SUBROUTINE cmcarp
```

Matrisler `GİRDİ` kütüğünden aşağıdaki şekilde temin edilirler

```
(1.,2.) (-1.,-1.) (0.,3.) (-4.,1.) (3.,2.)
(-1.,5.) (3.,-4.) (0.,1.) (-1.,0.)
(-1.,-3.) (1.,5.) (4.,3.) (-2.,3.) (8.,-6.)
(0.,4.) (-1.,6.) (3.,1.) (-3.,-7.)
```

Çıktı ise, `Sanal_Matris_Yazdir` alt programında 10 ile verilen formatta, aşağıda gibi verilmiştir.

```
A matrisi
(   1.0,   2.0)  (  -1.0,  -1.0)  (   0.0,   3.0)
(  -4.0,   1.0)  (   3.0,   2.0)  (  -1.0,   5.0)
(   3.0,  -4.0)  (   0.0,   1.0)  (  -1.0,   0.0)

B matrisi
(  -1.0,  -3.0)  (   1.0,   5.0)  (   4.0,   3.0)
(  -2.0,   3.0)  (   8.0,  -6.0)  (   0.0,   4.0)
(  -1.0,   6.0)  (   3.0,   1.0)  (  -3.0,  -7.0)

C matrisi
(  -8.0,  -9.0)  ( -26.0,  14.0)  (  23.0,  -2.0)
( -34.0,   5.0)  (  19.0,  -7.0)  (  11.0,  -4.0)
( -17.0, -13.0)  (  26.0,  18.0)  (  23.0,   0.0)

D matrisi
(   0.0,  -1.0)  (   0.0,   4.0)  (   4.0,   6.0)
(  -6.0,   4.0)  (  11.0,  -4.0)  (  -1.0,   9.0)
(   2.0,   2.0)  (   3.0,   2.0)  (  -4.0,  -7.0)
```

ÖRNEK 4: Bir $n \times n$ bilinmeyenli kompleks lineer denklem sistemini çözmek için bir altprogram yazınız ve aşağıdaki şekilde verilen sistemin çözümünü bulunuz.

$$\mathbf{A} = \begin{pmatrix} 1+2i & -1-i & 3i \\ -4+i & 3+2i & -1+5i \\ 3-4i & i & -1 \end{pmatrix} \qquad \mathbf{R} = \begin{pmatrix} -1-3i \\ 1+5i \\ 4+3i \end{pmatrix}$$

ve denklem sisteminin

$$\mathbf{AX} = \mathbf{R}$$

şeklinde verildiği kabul edilmektedir.

Burada uygulanan çözüm tekniği Bölüm13'de bahsedilen Gauss-Jordan yoketme tekniğidir; sadece çarpma, toplama ve bölme işlemleri kompleks sayılar ile yapılmaktadır. Bu programı daha iyi kavramak için Gerçek lineer denklem sistemlerine uygulanan Gauss-Jordan programı ile kıyaslayınız.

```
PROGRAM Ornek4
IMPLICIT NONE
COMPLEX, DIMENSION(3) :: r=(/(-1.,-3.),(1.,5.), &
                        (4.,3.)/), x
COMPLEX, DIMENSION(3,3):: a=RESHAPE( (/ (1.,2.),&
    (-4.,1.),(3.,-4.), (-1.,-1.),(3.,2.),&
    (0.,1.),(0.,3.),(-1.,5.),(-1.,0.) /), (/3,3/))
INTEGER :: i,j, n=3
OPEN(3,FILE='Cikti.dat',STATUS='unknown')
! Matrisi yazdır
DO i=1,n
   WRITE(3,10) (a(i,j),j=1,n),i,r(i)
END DO
! Gauss Yoektme Metodunu kompleks sayılara uygula
CALL CGauss_Yoketme_Metodu(n,a,r,x)
WRITE(3,15)
DO i=1,n
   WRITE(3,20) i,x(i) ! Çözümü yazdır
END DO
10 FORMAT(1x,'[',3(' (',F4.1,',',F4.1,')'),'] [ x(',&
   i1,') ]= [ (',F4.1,','F4.1,') ]')
15 FORMAT(/1x,'Denklem Sisteminin Çözümü'/)
20 FORMAT(1x,' x(',i2,') = ',F9.5,'+',F9.5,' i')
END PROGRAM Ornek4

SUBROUTINE CGauss_Yoketme_Metodu(n,a,r,x)
IMPLICIT NONE
INTEGER, INTENT(IN) :: n
COMPLEX, DIMENSION(n), INTENT(INOUT)  :: r, x
COMPLEX, DIMENSION(n,n), INTENT(INOUT):: a
COMPLEX :: toplam, p
INTEGER :: i, j, k
!
!  Gauss Yoketme Metodu ile sanal katsayılı Lineer
```

```
     !  Denklem Sisteminin Çözümünü verir.
     !  (n x n)   ŞEKLİNDEKİ LİNEER DENKLEM SİSTEMİNİN
     !   ** a(n,n) : DENKLEM SİSTEMİNİ BELİRLEYEN MATRİS
     !   ** r(n)   : SAĞ TARAF VEKTÖRÜ
     !   ** x(n)   : BİLİNMEYENLER VEKTÖRÜDÜR.
     !  UYARI!  Alt programa girilen matris ve vektörlerin
     !          başlangıç değerleri bozulmaktadır...
     DO k=1,n
        p=1.0/a(k,k)
        r(k)=r(k)*p
         DO i=k+1,n
            a(k,i)=a(k,i)*p
         END DO
         DO i=k+1,n
            DO j=k+1,n
               a(i,j)=a(i,j)-a(i,k)*a(k,j)
            END DO
            r(i)=r(i)-a(i,k)*r(k)
         END DO
     END DO
     x(n)=r(n)
     DO k=n-1,1,-1
        toplam=0.0
        DO j=k+1,n
           toplam=toplam+a(k,j)*x(j)
        END DO
        x(k)=r(k)-toplam
     END DO
     END SUBROUTINE CGauss_Yoketme_Metodu
```

```
[ ( 1.0, 2.0) (-1.0,-1.0) ( 0.0, 3.0)] [ x(1) ]= [ (-1.0,-3.0) ]
[ (-4.0, 1.0) ( 3.0, 2.0) (-1.0, 5.0)] [ x(2) ]= [ ( 1.0, 5.0) ]
[ ( 3.0,-4.0) ( 0.0, 1.0) (-1.0, 0.0)] [ x(3) ]= [ ( 4.0, 3.0) ]

Denklem Sisteminin Çözümü

 x( 1) =   0.55684+  0.95495 i
 x( 2) =   2.14574+  1.84847 i
 x( 3) =  -0.35814+ -0.21678 i
```

ALIŞTIRMALAR

15.1 Herhangi bir kompleks sayının n.ci dereceden kuvvetini De Moivre formülü uyarınca hesaplayan bir program yazınız. De Moivre formülü

$$\left[r\left(\cos\theta+i\sin\theta\right)\right]^n = r^n\left(\cos n\theta+i\sin n\theta\right)$$

olarak verilmektedir.

15.2 Herhangi $n\times n$ boyutlu bir **A** kompleks matrisinin tersini Gauss-Jordan metodu ile hesaplayan bir program yazınız.

15.3 Herhangi bir matrisin tersini ve kendisim sağ taraftan veya sol taraftan çarparak birim matrisin verip vermediğini kontrol eden bir program yazınız.

15.4 Bir $z=a+ib$ kompleks sayısının doğal logaritmasını hesaplayan bir altprogram yazınız. NOT: Burada $a,b\neq 0$ için

$$\ln\left(a+ib\right)=\ln\sqrt{a^2+b^2}+i\theta$$

15.5 Bir $z=a+ib$ kompleks sayısının üstel değerini, yani $\omega=e^z$, hesaplayan bir alt program yazınız. NOT:

$$e^z=\exp\left(a+ib\right)=e^a\left(\cos b+i\sin b\right)$$

15.6 Bir $z=a+ib$ kompleks sayısının trigonometrik değerlerini hesaplayan birer altprogram yazınız. NOT:

$$\cos\left(a+ib\right)=\cos\left(a\right)\cosh\left(b\right)-i\sin\left(a\right)\sinh\left(b\right)$$
$$\sin\left(a+ib\right)=\sin\left(a\right)\cosh\left(b\right)+i\cos\left(a\right)\sinh\left(b\right)$$
$$\tan\left(a+ib\right)=\sin\left(a+ib\right)/\cos\left(a+ib\right)$$
$$\sec\left(a+ib\right)=1/\cos\left(a+ib\right)$$

formülleriyle yapılan işlemin sonuçlarını kontrol ediniz.

15.7 $ax^2+bx+c=0$ ile verilen ikinci dereceden denklemin köklerini bulmak için en genel bir program yazınız. *NOT:* `COMPLEX` değişkenlerden yararlanınız.

15.8 Katsayıları, a , b ve c, birer kompleks sayı olan ikinci dereceden denklemin kompleks köklerini hesaplayan bir altprogram yazınız. Not:

$$\sqrt{x+iy}=\sqrt{r}\left[\cos\left(\frac{\theta}{2}\right)+i\sin\left(\frac{\theta}{2}\right)\right]$$

Burada $r=\sqrt{x^2+y^2}$ ve $\theta=\arctan\left(y/x\right)$ olarak alınacaktır.

15.9 Aşağıdaki integralleri yamuklar ve Simpson kuralı ile hesaplayan bir program yazınız.

(a) $\int_{1-2i}^{-1+i} z^2\sin z dz$ (b) $\int_{0}^{3-2i} e^{-z^2}dz$ (c) $\int_{-i}^{1+3i}\cos z^2 dz$ (d) $\int_{-1}^{2-i} e^{\sin z}dz$

15.10 Aşağıdaki seri ifadeleri hesaplayan bir program yazınız. Programı durdurma kriteri olarak $\left|z_{n+1}\right|<\varepsilon$ kullanınız. Unutmayınız ki burada kullanılan mutlak değer ifadesi kompleks sayının büyüklüğüdür.

(a) $\sum_{n=0}^{\infty}\frac{\left(1+2i\right)^n}{n!}$ (b) $\sum_{n=0}^{\infty}\frac{\left(3-2i\right)^n}{n^n}$ (c) $\sum_{n=0}^{\infty}\frac{\left(3n+2i-1\right)^n}{2^n n!}$ (d) $\sum_{n=0}^{\infty}\frac{\left(i+1\right)^{2n+1}}{\left(2n+1\right)!}$

15.11 Aşağıdaki seri ifadeleri dışarıdan temin edilen herhangi bir z kompleks sayısı için hesaplayan bir program yazınız. Programı durdurma kriteri olarak aynı mantıktan hareket ediniz.

(a) $\sum_{n=0}^{\infty} n\left(\frac{z}{3}\right)^n$ (b) $\sum_{n=0}^{\infty} \frac{\left(z-3+2i\right)^n}{2^n}$ (c) $\sum_{n=0}^{\infty} \frac{z\left(3n-1\right)}{3^n\left(3n-1\right)!}$ (d) $\sum_{n=0}^{\infty} \frac{n^n}{n!}\left(z-1\right)^n$

15.12 Bir $n \times n$ **A** kompleks matrisi için, sin**A** değerini Taylor serisi açılımı yardımıyla hesaplayan bir program yazınız. Taylor serisi

$$\sin \mathbf{A} = \mathbf{A} - \frac{1}{3!}\mathbf{A}^2 + \frac{1}{5!}\mathbf{A}^5 + \cdots$$

olarak veriliyor. Programda yakınsama kriteri olarak *n*.ci terim için

$$\frac{1}{\left(2n-1\right)!}\max\left|\mathbf{A}\right|^{2n+1} < \varepsilon$$

kriterini kullanınız; yani **A** kompleks matrisinin mutlak değerce en büyük elemanının kuvveti alınacaktır.